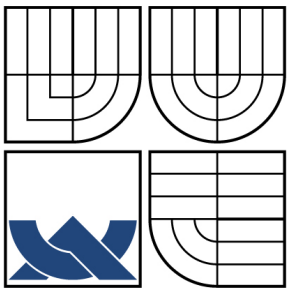
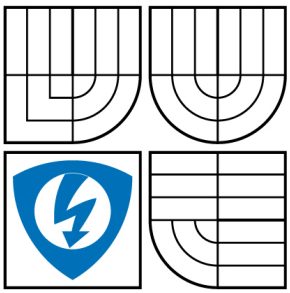


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF CONTROL AND INSTRUMENTATION

MRAVENČÍ KOLONIE

ANT COLONY

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. PAVEL HART

AUTHOR

VEDOUCÍ PRÁCE — Ing. PETR HONZÍK, Ph.D.

SUPERVISOR

BRNO 2008

VYSOKÉ UČENÍ
TECHNICKÉ V BRNĚ
Fakulta elektrotechniky
a komunikačních technologií
Ústav automatizace a měřicí techniky

Diplomová práce

magisterský studijní obor
Kybernetika, automatizace a měření

Student: Hart Pavel, Bc.
Ročník: 3

ID: 22986
Akademický rok: 2007/08

NÁZEV TÉMATU:

Mravenčí kolonie

POKYNY PRO VYPRACOVÁNÍ:

Cílem diplomové práce je provést rešerši optimalizačních algoritmů a dvě metody naprogramovat. Následně se seznámit s algoritmy typu mravenčí kolonie, naprogramovat je a provést srovnání na konkrétních příkladech, vyhodnotit dosažené výsledky

DOPORUČENÁ LITERATURA:

Dle doporučení školitele.

Termín zadání: 3.12.2007 ***Termín odevzdání:*** 26.5.2008

Vedoucí projektu: Ing. Petr Honzík, Ph.D.

prof. Ing. Pavel Jura, CSc.
předseda oborové rady

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ Fakulta elektrotechniky a komunikačních technologií
Ústav automatizace a měřicí techniky

UPOZORNĚNÍ:

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení: Bc. Pavel Hart
Bytem: Otavská 1067/9, 37011, České Budějovice - České Budějovice 2
Narozen/a (datum a místo): 5.2.1983, České Budějovice

(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta elektrotechniky a komunikačních technologií
se sídlem Údolní 244/53, 60200 Brno 2
jejímž jménem jedná na základě písemného pověření děkanem fakulty:
doc. Ing. Václav Jirsík, CSc.

(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   - ☐ disertační práce
   - ☑ diplomová práce
   - ☐ bakalářská práce

   jiná práce, jejíž druh je specifikován jako .....

(dále jen VŠKP nebo dílo)

Název VŠKP: Mravenčí kolonie
Vedoucí/školitel VŠKP: Ing. Petr Honzík, Ph.D.
Ústav: Ústav automatizace a měřicí techniky
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:

- ☑ tištěné formě - počet exemplářů 1
- ☑ elektronické formě - počet exemplářů 1

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.

4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti
   - ☑ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Vysoké učení technické v Brně
Fakulta elektrotechniky a komunikačních technologií
Ústav automatizace a měřící techniky

Mravenčí kolonie

Diplomová práce

Obor: Kybernetika, automatizace a měření
Autor práce: Bc. Pavel Hart
Vedoucí práce: Ing. Petr Honzík, Ph.D.

Abstrakt:

Práce se zabývá rešerší optimalizačních algoritmů a dále pak implementací a porovnáním tří z nich. Jedná se o algoritmus mravenčí kolonie, zakázané prohledávání a simulované žíhání. Implementace algoritmů byla uzpůsobena k řešení problému obchodního cestujícího. U všech zmíněných algoritmů byla zkoumána a zhodnocena jejich časová náročnost a kvalita nalezeného řešení. U algoritmů mravenčí kolonie bylo navíc provedeno zhodnocení vlivu řídicích parametrů na kvalitu nalezeného řešení.

Co do časové náročnosti výpočtu se nejlepším ukázal algoritmus zakázaného prohledávání (Tabu Search). Algoritmy mravenčí kolonie (ACO) a simulované žíhání (Simulated Annealing) byly v tomto ohledu rovnocenné. Nejkvalitnějších řešení (nejkratších cest) dosáhl algoritmus mravenčí kolonie.

Klíčová slova: Mravenčí kolonie, ACO, Optimalizační algoritmy, Zakázané prohledávání, Simulované žíhání, Problém obchodního cestujícího, TSP, implementace, srovnání algoritmů.

Brno University of Technology
Faculty of Electrical Engineering and Communication
Department of Control, Measurement and Instrumentation

Ant Colony

Master's thesis

Specialization of study: Cybernetics, Control and Measurement
Author: Bc. Pavel Hart
Supervisor: Ing. Petr Honzík, Ph.D.

Abstract:

First part of the thesis is about literature research of optimization algorithms. Three of the algorithms were implemented and tested, concretely the ant colony algorithm, tabu search and simulated annealing. All three algorithms were implemented to solve the traveling salesman problem. In second part of the thesis the algorithms were tested and compared. In last part the influence of the ant colony parameters was evaluated.

Tabu Search algorithm turned out to be computationally least time consuming. Ant colony algorithm and simulated annealing was equal in this respect. The best results (shortest ways) were produced by ant colony algorithm.

Key words: Ant colony algorithm, ACO, Optimization algorithms, Tabu search, Simulated annealing, Traveling salesman problem, TSP, implementation, Algorithms comparison.

Bibliografická citace

HART, P. *Mravenčí kolonie.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. 61 s. Vedoucí diplomové práce Ing. Petr Honzík, Ph.D.

P r o h l á š e n í

„Prohlašuji, že svou diplomovou práci na téma Mravenčí kolonie jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.“

V Brně dne : Podpis:

P o d ě k o v á n í

Děkuji tímto svému vedoucímu práce za cenné připomínky a rady při vypracování diplomové práce. Dále děkuji své partnerce za morální podporu a za výpomoc při gramatické kontrole této práce, svým rodičům za trpělivost a finanční podporu mého studia. Nakonec bych též rád poslal malé poděkování hudebním skupinám Rammstein a Kabát, jejichž rázné písně s mravenčí tématikou mi dodávaly chuť do práce.

V Brně dne : Podpis:

OBSAH

SEZNAM OBRÁZKŮ A GRAFŮ

SEZNAM TABULEK

1. ÚVOD

Řešení problému by trvalo milión let a přesto je třeba mít výsledek do zítra? Je to možné a když ano tak jak? Odpovědí na tyto otázky je optimalizace. Používá se všude tam, kde by bylo řešení problému analytickými metodami silně nevhodné či dokonce nemožné.

Většina problémů inženýrské praxe může být řešena jako optimalizační problém. Postup je takový, že řešený problém se nejprve převede na problém matematický daný vhodným funkčním předpisem. Jeho optimalizace pak vede k nalezení argumentů tzv. účelové funkce (v podstatě se snažíme najít minimum či maximum).

V první kapitole je uvedeno rozdělení optimalizačních algoritmů a popis nejznámějších z nich. Dva z těchto algoritmů jsou implementovány, a to k řešení problému obchodního cestujícího. Poté se práce podrobněji zabývá optimalizací pomocí mravenčích kolonií, což je algoritmus inspirovaný chováním skutečných biologických mravenců. Algoritmus je implementován pro řešení stejného problému jako předchozí dva a všechny tři jsou porovnány z jak hlediska časové náročnosti, tak i z hlediska kvality poskytovaného řešení.

Je třeba poznamenat, že názvy některých algoritmů a problémů, které jsou v anglické literatuře běžně zavedené, jsou ponechány v originálním znění (to se týká i některých zkratek), popřípadě pokud jsou přeloženy, je jejich anglický ekvivalent uveden v závorce.

2. OPTIMALIZAČNÍ ALGORITMY

2.1 TYPY OPTIMALIZAČNÍCH ALGORTIMŮ

Algoritmů sloužících k optimalizaci je několik typů. Jedno z možných rozdělení je uvedeno v následující tabulce.

Tabulka 2.1 Rozdělení optimalizačních algoritmů

Deterministické	Stochastické	Smíšené
Greedy Algorithm	Random Search-walk	Ant Colony Optimalization
Hill Climbing	Simulated Annealing	Immune System Methods
Branch & Bound	Monte Carlo Method	Memetic Algorithms
Depth-first Search	Tabu Search	Scatter Search & Path Relinking
Best-first Search	Evolutionary Computation	Particke Swarm
	Stochastic Hill Climbing	Genetic Algorithms
		SOMA (Self Organizing Migrating Algorithm)

2.1.1 Enumerativní

Metoda spočívá ve výpočtu všech možných kombinací daného problému (tudíž se nejedná o optimalizační algoritmy). Tento přístup je vhodný jen pro problémy, u nichž jsou argumenty účelové funkce diskrétního charakteru a nabývají malého počtu hodnot. V opačném případě by výpočet mohl trvat i milióny let.

2.1.2 Deterministické

Tyto algoritmy jsou postavené pouze na rigorózních metodách klasické matematiky. Pro dosažení efektivních výsledků obvykle vyžadují následující předpoklady:

- Lineárnost problému
- Konvexnost problému
- Malý prohledávaný prostor možných řešení
- Spojitost prohledávaného prostoru možných řešení
- Účelová funkce by měla mít pokud možno pouze jeden extrém
- Mezi parametry „uvnitř“ účelové funkce se nevyskytují nelineární interakce

- Jsou dostupné informace typu gradient apod.
- Problém je definován v analytickém tvaru

Výhodou pak je, že výsledkem algoritmu je pouze jedno řešení.

2.1.3 Stochastické

Stochastické algoritmy jsou založeny na využití náhody. V podstatě se jedná o čistě náhodné hledání hodnot argumentů účelové funkce. Výsledkem je pak to nejlepší řešení, které bylo nalezeno během celého hledání.

Stochastické algoritmy bývají pomalé a jsou vhodné pro prohledávání malých prostorů možných řešení a jsou tak vhodné především pro hrubý odhad.

2.1.4 Smíšené

Jedná se o algoritmy, jež jsou postaveny na kombinacích deterministických a stochastických metod. Dosahují velmi dobrých výsledků a jsou vhodné u problémů s velkým prostorem možných řešení [1][5].

2.2 STRUČNÝ POPIS VYBRANÝCH ALGORITMŮ

2.2.1 Hladový algoritmus (Greedy Algorithm)

Hladový algoritmus řeší problém hledáním lokálního optima v každém kroku a snaží se tak najít optimum globální. Iterativně se provádí jedna hladová volba (Greedy choice) za druhou a řešený problém se tak redukuje na menší problémy.

Například při řešení problému obchodního cestujícího pracuje hladový algoritmus takto: „V každém kroku navštiv nejbližší nenavštívené město".
Algoritmus nejlépe funguje na problémech následujících vlastností:

- **Hladová rozhodovací vlastnost** (Greedy choice property): Můžeme udělat jakoukoli volbu, která se zdá v daném okamžiku nejlepší a vyvstalé podproblémy řešit později. Volba může záviset na volbách předchozích. Nemůže být však závislá na volbách, které budou následovat, ani na řešení

všech podproblémů. Hladový algoritmus nikdy znovu neuvažuje vlastní volby.

- **Problém má optimální substrukturu** (optimal substructure): Problém vykazuje optimální substrukturu, pokud se optimální řešení problému skládá z optimálních řešení podproblémů.

Pro velkou řadu ostatních problémů může hladový algoritmus poskytnout i nejhorší možné řešení problému. Příkladem může být výše zmíněný problém obchodního cestujícího.

Přestože velmi často nedokáže hladový algoritmus nelézt optimální řešení, je užitečný, protože je rychlý a často dává dobrou aproximaci optima [5].

2.2.2 Horolezecký algoritmus (Hill Climbing)

Princip funkce horolezeckého algoritmu je snadno pochopitelný z následujícího pseudokódu.

```
1   x = náhodně vygenerovaný vektor;
2   time = 0;
3   f_min = ∞;
4   while (time < time_max){
5         time = time + 1;
6         f(x*) = min f(x'), kde x'∈U(x);
7         if f(x*) < f_min {
8               f_min = f(x*);
9               x_min = x*;
10        }
11        x = x*;
12  }
```

Tabulka 2.2 Popis proměnných algoritmu

f_{min} x_{min}	Záznam nejlepšího řešení
time	Iterace
$time_{max}$	Počet iterací
x*	Řešení
x'	Aktuální řešení
U(x)	Okolí bodu (řešení) x

Algoritmus je jednoduchý, ale má mnoho nevýhod. Největší nevýhodou je možnost nalezení pouze lokálních optim (minimum či maximum). Dále se mohou vyskytnout následující problémy.

- **Problém zacyklení:** Po konečném počtu iteračních kroků se algoritmus vrací k řešení, které se již vyskytlo v některém z předešlých kroků, přičemž nejlepší zaznamenané řešení je vzdálené od globálního optima funkce.
- **Hřeben** (Ridges): Hřeben je křivka vedoucí k maximu. Každý bod na hřebenu se tváří jako lokální maximum, i když je bod součástí křivky vedoucí k lepšímu optimu.
- **Plošina** (Plateau): Nastává v plochých částech prohledávání prostoru. V těchto místech pak dostáváme cestu, kde jsou všechny heuristiky velmi blízko u sebe a my cestujeme bezcílně po okolí[2][5].

2.2.3 Stochastický horolezecký algoritmus (Stochastic Hill Climbing)

Jedná se o modifikaci horolezeckého algoritmu. Ta spočívá v tom, že horolezecký algoritmus zkoušíme aplikovat vícekrát a náhodně měníme výchozí bod výpočtu. Cílem této modifikace je snaha o omezení možnosti uváznutí v lokálním optimu funkce [2][5].

2.2.4 Zakázané prohledávání (Tabu Search)

Zakázané prohledávání vychází z horolezeckého algoritmu a snaží se odstranit problém zacyklení tím, že do něj zavádí krátkodobou paměť. Ta si po určitý čas pamatuje inverzní transformace k lokálně optimálním transformacím řešení použitých k získání nových "středů" pro jednotlivé iterace. Tyto inverzní transformace jsou při tvorbě nového okolí zakázány.

Některé verze algoritmu mají vedle paměti krátkodobé navíc i paměť dlouhodobou. Ta má za účel znevýhodňovat transformace, které se sice nenacházejí v krátkodobé paměti, přesto se však již během zpracování vyskytly.

Velmi důležitým parametrem, majícím vliv na kvalitu výsledků algoritmu, je velikost zakázaného seznamu (k). Pokud je k příliš malé, pak se může vyskytnout

zacyklení. Pokud je však příliš velké, je zde možnost, že při prohledávání je „přeskočeno“ některé hluboké údolí funkce *f(x)* (tzn. přicházíme o nadějné lokální minimum, které se může později ukázat být globálním). Algoritmus je možno modifikovat tak, aby se během výpočtu délka zakázaného seznamu dynamicky měnila [2].

```
13 x = náhodně vygenerovaný vektor;
14 time = 0;
15 f_min = ∞;
16 T = 0;
17 while (time < time_max){
18       time = time + 1;
19       f_loc - min = ∞;
20       for (t∈S){
21             x' = tx;
22             if ((f(x`) < f_loc - min )AND((t∉T)OR(f(x`) < f_min)){
23                   x* = x`;
24                   t* = t;
25                   f_loc - min = f(x`);
26             }
27       }
28       if (f_loc - min < f_min ){
29             f_min = f_loc - min;
30             x_min = x*;
31       }
32       x = x*;
33       if (|T| < k){
34             T = T ∪ {t*^-1};
35       }else{
36             T = (T ∪ {t*^-1})/{t̂};
37       }
38 }
```

Tabulka 2.3 Popis proměnných algoritmu

f_{min}	Záznam nejlepšího řešení
x_{min}	
time	Iterace
$time_{max}$	Počet iterací
f_{loc}	Lokálně optimální řešení
x*	
x'	Nové řešení
T	Seznam zakázaných transformací
S	Seznam transformací
k	Maximální velikost T

2.2.5 Větvení a meze (Branch and Bound)

Algoritmus obohacuje enumerativní přístup k řešení problému o možnost ořezat (a tedy při výpočtu dále neuvažovat) méně slibné části prostoru možných řešení. Algoritmus se skládá ze dvou procedur, a to z větvení (branching) a poutání (bounding).

- **Větvení:** Tato procedura se snaží rozdělit oblast možných řešení na podoblasti. Tento postup se aplikuje rekurzivně a vzniká tak stromová struktura, jejímiž uzly jsou vytvořené podoblasti.
- **Meze**: Procedura rychle hledající horní a dolní limit pro optimální řešení uvnitř regionu.

Hlavní princip je, že pokud nižší mez pro podoblast A z prohledávání stromu je větší než horní mez pro kteroukoli z předešle zkoumaných podoblastí B, pak A může být bezpečně odstraněno z vyhledávání. Tato část algoritmu se nazývá prořezávání.

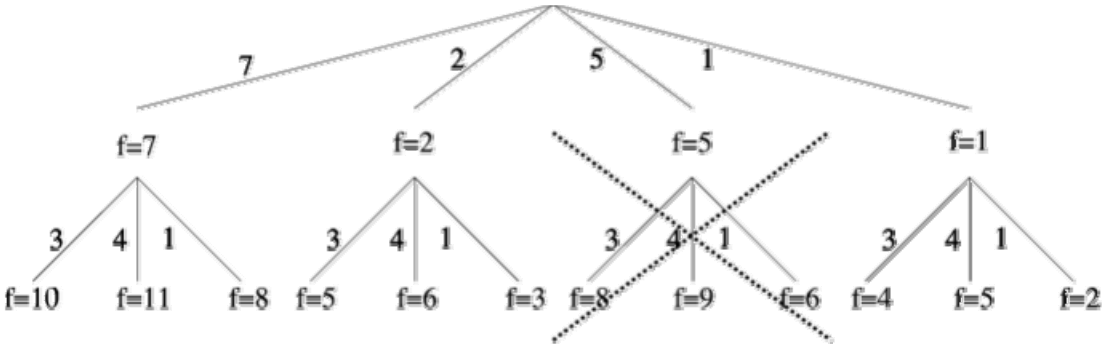
Pokud se dolní a horní mez uzlu rovnají, pak se uzel nazývá vyřešený.

Algoritmus končí, pokud jsou všechny uzly stromu buď prořezané, nebo vyřešené. V tom momentě minimální dolní mez a maximální horní mez určují interval, na kterém leží globální optimum.

Metoda větvení a mezí je názorně demonstrována na následujícím příkladu. Předpokládejme optimalizační problém, jehož řešení je popsáno vektorem $a=(a_1,a_2)$, přičemž a_1 je z {7, 2, 5, 1}, a_2 je z {3, 4, 1}. Hodnota kriteriální funkce je dána vztahem $f = a_1 + a_2$. Nejlepší dosud nalezená hodnota f je zde označována f_b [1][5][7].

Tabulka 2.4 Příklad použití metody Větvení a mezí

Krok	a	f	f_b	pozn.
1	-7,3	10	10	
2	-7,1	8	8	
3	-7,4	11	8	
4	(2,x)	2+x	8	$f < f_b$, pokračuje se
5	-2,3	5	5	
6	-2,1	3	3	
7	-2,4	6	3	
8	(5,x)	5+x	3	$f > f_b$, nepokračuje se
9	(1,x)	1+x	3	$f < f_b$, pokračuje se
10	-1,3	4	3	
11	-1,4	5	3	
12	-1,1	2	2	



Obrázek 2.1 Příklad použití metody Větvení a mezí [7]

2.2.6 Simulované žíhání (Simulated Annealing)

Simulované žíhání je algoritmus inspirovaný žíháním tuhého tělesa, které slouží k odstraňování vad krystalové mřížky. Krystal se zahřeje na vysokou teplotu a poté se pomalu ochlazuje (žíhá). V simulovaném žíhání je krystal reprezentován binárním řetězcem x a energie funkční hodnotou *f(x)*. Při žíhání se minimalizuje energie krystalu a tedy funkce *f(x)*. Podrobně je algoritmus popsán níže.

```
1  x = náhodně vygenerovaný binární řetězec;
2  T = Tmax;
3  X* = x;
4  k = 1;
5  while ((T > Tmin) AND (k >0){
```

```
6         t = t + 1;
7         x' = transformace(x);
8         if ((f(x`) ≤ fx){
9               Pr = 1;
10        }else{
11        Pr = exp(-(f(x`)-f(x))/T);
12        }
13        if (random < Pr ){
14              x = x`;
15              k = k+1;
16              if (f(x) < f(x*)){
17                    x* = x;
18              }
19        }
20 T = α·T;
21 }
```

Tabulka 2.5 Popis proměnných algoritmu

Pr	Pravděpodobnost nahrazení stávajícího řetězce novým
x*	Řešení
x'	Nové řešení
T	Teplota
α	Koeficient snižování teploty
k	Počet úspěšných pokusů

Proces snižování teploty se nalézá na řádku 14. Konstanta α je menší než 1 (obvykle bývá 0,9)[2].

2.2.7 Evoluční strategie (Evolutionary Computation)

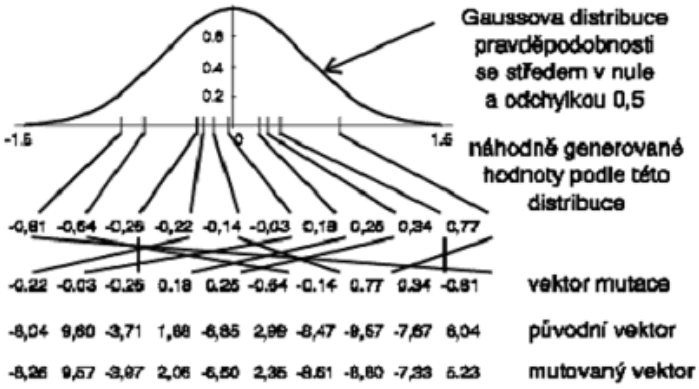
Evoluční strategie vychází z všeobecných představ přirozeného výběru, není však tak komplexní jako genetické algoritmy. Základem je předpis, který mutuje aktuální řešení *x* na nové řešení *x'*. Pokud nové řešení je lepší než předchozí, postupuje do další generace. Názorně je to vidět na následujícím zápisu a obrázku [2].

```
1  x = náhodně vygenerovaný vektor reálných proměnných;
2  t = 0;
3  σ = σini;
4  x* = x;
5  while (t < tmax){
6        i = 0;
7        k = 0;
8        while (i < imax){
9              i = i + 1;
10             if (f(x`) < f(x)){
11                   k = k + 1;
```

```
12                    x = x`;
13              }
14              if (f(x) < f(x*)){
15                    x* = x;
16              }
17
18        }
19        if (k/i_max < 0.2 ){
20              σ = c_d ·σ;
21        }else{
22              if (k/i_max > 0.2){
23                    σ = c_i· σ;
24              }
25        }
26  }
```

Tabulka 2.6 Popis proměnných algoritmu

Pr	Pravděpodobnost nahrazení stávajícího řetězce novým
x*	Řešení
x'	Nové řešení
t	Čítač epoch
i_{max}	Počet cyklů pro danné σ
σ	Směrodatná ochylka
k	Počet úspěšných pokusů
ci	Koeficienty zmenšování σ
cd	



Obrázek 2.2 Příklad mutace vektoru za užití Gaussova rozložení [2]

2.2.8 Memetické algoritmy (Memethic Algorithms)

Memetické algoritmy jsou založeny na populačním přístupu k heuristickému prohledávání. Někdy jsou označovány též jako hybridní genetické algoritmy či

paralelní genetické algoritmy. Jedná se vlastně o kombinaci genetického algoritmu s lokálním prohledáváním.

Pro některé typy problémů vykazují oproti genetickým algoritmům větší efektivitu. Jsou používány například pro predikci proteinových struktur [5].

2.2.9 Rozptýlené prohledávání (Scatter Search)

Rozptýlené vyhledávání pracuje s množinou řešení (referenční množina) tak, že z ní vždy dvě řešení vybere a lineární kombinací z nich vytvoří řešení nové [8].

2.2.10 Optimalizace pomocí částicového roje (Particle swarm optimization - PSO)

Tento algoritmus je inspirován chováním živočichů sdružujících se do rojů či hejn. Jako příklad lze uvést chování hejna ryb. Když jedna ryba nalezne hledanou cestu (jídlo, útočiště atp.), zbytek ryb v hejnu bude rychle cestu následovat, a to i když se nacházejí na zcela opačné straně hejna.

V samotném algoritmu se pracuje s rojem částic poletujících v multidimenzionálním prostoru možných řešení. Každá částice si pamatuje nejen pozici nejlepšího řešení, které sama nalezla, ale má i vědomost o pozici nejlepšího řešení nalezeného celým rojem. Částice si totiž navzájem předávají informace o dobrých pozicích a upravují svou pozici a rychlost závisle na těchto dobrých informacích. Částice jsou tahány jak k vlastní nalezené nejlepší pozici, tak i k nejlepší pozici nalezené celým rojem. Komunikace mezi částicemi roje probíhá dvěma způsoby:

- Globální nejlepší pozice, která je známa všem a je okamžitě aktualizována, pokud jakákoli částice najde novou nejlepší pozici
- „Sousední“ nejlepší pozice, kde každá částice komunikuje pouze s podmnožinou roje o nejlepší pozici

Sousední nejlepší pozice dovoluje lepší prozkoumávání prostoru a snižuje citlivost PSO na uváznutí v lokáním minimu, avšak snižuje rychlost konvergence [5].

2.2.11 Rozdílová evoluce (Differetial Evolution)

Tento typ optimalizačního algoritmu patří do třídy evolučních strategií. Princip je nejlépe vidět na následujícím algoritmu [5, 9].

```
1  do{
2        for(každý rodič Xc v X){
3              Náhodně vyber rozdílový pár Xa a Xb;
4              Vypočítej  X'c = Xc + F × (Xa − Xb);
5              Vypočítej  X''c = Crossover(X'c, Xc);
6              Urči kvalitu X``c;
7              if (X``c je lepší než Xc){
8                    X``c jde do nové populace;
9              }else{
10                   Xc  jde do nové populace;
11             }
12       }
13       X = X`;
14 }while (populace konverguje nebo byl dosažen populační limit)
```

Tabulka 2.7 Popis proměnných algoritmu

X	Populace
X'	Nová populace
Xa	Náhodné vektory
Xb	
Xc	Rodičovský vektor

2.2.12 Genetické algoritmy (Genetic Algorithms)

Genetický algoritmus je heuristický přístup řešící problém aplikací evoluční biologie a patří mezi evoluční algoritmy. Princip spočívá v postupné tvorbě generací řešení. Postupem generací se řešení zlepšuje. Využívá se zde procesů jako je dědičnost, mutace, přirozený výběr a křížení. Algoritmus je uveden níže [2][10].

```
1  P0 = náhodně vygenerovaná populace chromozómů;
2  t = 0;
3  σ = σini;
4  Ohodnoť každý chromozóm z populace P0 silou;
5  while (t < tmax){
6        t = t + 1;
7        Q = kvazináhodně vybrané dvojice chromozómů z Pt - 1
   s největší silou pomocí rulety;
8        Q =  Operace křížení(Q);
9        Q = Mutace jednotlivých chromozómů(Q);
10       Ohodnoť každý chromozóm z Q silou;
11       R = kvazináhodně vybrané chromozómy z Pt - 1 s nejmenší
   silou;
12       Pt = ((Pt - 1)/ R) ∪ Q;
```

Tabulka 2.8 Popis proměnných algoritmu

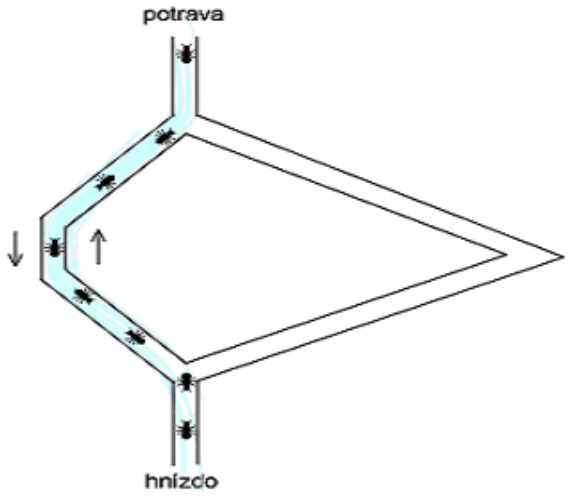
x	Chromozomy
x'	
t	Čítač epoch
P	Populace
Q	Subpopulace
R	

3. MRAVENČÍ KOLONIE

Optimalizace pomocí mravenčí kolonie neboli ACO (Ant Colony Optimalization) je jedním z moderních přístupů k řešení problému optimalizace při řešení složitých úloh (rozvoj od devadesátých let minulého století). ACO využívá stejně jako PSO rojovou inteligenci, v tomto případě kolonie mravenců. Tento typ algoritmů byl s úspěchem aplikován na řadu náročných optimalizačních úloh.

3.1 INSPIRACE PŘÍRODOU

Metoda ACO je inspirována komunikačními mechanizmy reálných mravenčích kolonií. Mravenci jsou drobní a téměř slepí živočichové žijící ve velkých, dobře organizovaných koloniích. Při hledání potravy se orientují především podle pomalu se odpařujících feromonových stop zanechávaných ostatními mravenci. Pokud nastane situace, kdy si má mravenec vybrat cestu, po které bude pokračovat, volí s větší pravděpodobností směr, ve kterém detekuje více feromonů. Sám též při tomto putování feromon za sebou zanechává. Od zdroje nalezené potravy se nejrychleji vrací mravenec, který zvolil nejkratší trasu k ní. Pozitivní zpětná vazba pak způsobí, že nejkratší cesta je volena mravenci čím dál tím častěji, až zcela převládne.



Obrázek 3.1 Binární most s nestejně dlouhými rameny [3]

Experimenty prováděné na mravencích dokázaly, že šance vybrání kratšího ramene se zvyšuje s poměrem délek obou ramen. Problém může nastat, pokud si více mravenců na počátku vybere delší rameno, nebo je kratší přidáno až po určitém čase. Pak může nastat situace, kdy mravenci neustále cestují k cíli delším ramenem. Avšak i na toto příroda našla řešení. Určitý druh mravenců totiž vykazuje zajímavou vlastnost. Pokud je mravenec v polovině dlouhého ramene, uvědomí si, že směřuje k cíli téměř kolmo, a tak se otočí a zamíří zpět [3][4].

3.2 UMĚLÍ MRAVENCI

Umělý mravenec je struktura vytvořená pro účely použití při implementaci optimalizačních algoritmů. Ačkoli je jeho chování inspirováno chováním reálných mravenců, v mnoha ohledech se liší. Odlišnosti jsou způsobené tím, že se nepokoušíme simulovat chování reálných mravenců, ale tímto chováním se chceme pouze inspirovat a řešit zadané úlohy rychle a kvalitně.

Podobnosti se skutečnými mravenci:

- Pracujeme s kolonií kooperujících mravenců
- Nepřímá komunikace mravenců je zprostředkovaná pomocí feromonu
- Mravenci používají pravděpodobnostní rozhodování, strategie rozhodování je lokální

Rozdíly oproti skutečným mravencům:

- Umělí mravenci se pohybují v diskrétním světě
- Umělí mravenci mají vnitřní stavy (krátkodobou paměť). Mravenec zaznamenává doposud vykonané akce
- Umělí mravenci nejsou zcela slepí, mají k dispozici lokální informace (zrak, tabu)
- Množství zanechaného feromonu je funkcí kvality nalezeného řešení
- Časování ukládání feromonu je problémově závislé
- K ukládání feromonu může docházet až po projití celé cesty [3][4]

3.3 ACO METAHEURISTIKA

Na následujících řádcích je uvedena ACO metaheuristika. ACO metaheuristikou se rozumí obecný postup popsaný níže. ACO algoritmem je pak její libovolnou instancí. Označení mravenčí algoritmus (Ant algorithm) se používá pro algoritmy, které sice využívají některé z uvedených principů, ale nemusí plně odpovídat ACO metaheuristice.

```
 1 procedure ACO_Metaheuristika() {
 2    while (ukoncovaci_kriteria_nesplnena) {
 3       generovani_a_aktivita_mravencu();
 4       odparovani_feromonu();
 5       demonice_akce(); //volitelne - vyuziti globalnich informaci
 6    }
 7 }
 8
 9 procedure generovani_a_aktivita_mravencu() {
10    while (dostupne_zdroje) {
11       naplanuj_vytvoreni_noveho_mravence();
12       novy_aktivni_mravenec();
13    }
14 }
15
16 procedure novy_aktivni_mravenec(); {
17    inicializuj_mravence();
18    M = aktualizuj_pamet_mravence();
19    while (momentalni_stav != cilovy_stav) {
20       A = nacti_lokalni_smerovaci_tabulku();
21       P = vypocti_pravdepodobnosti_prechodu(A, M,
omezujici_podm);
22       nasledujici_stav = aplikuj_rozhodovaci_strategii(P,
omezujici_podminky);
23       presun_do_stavu(nasledujici_stav);
24       if (aktualizace_feromonu_krok-po-kroku) {
25          uloz_feromon_na_navstivene_hrany();
26          aktualizuj_smerovaci_tabulu();
27       }
28       M = aktualizuj_vnitrni_stav();
29    }
30    if (aktualizace_feromonu_zpozdena) {
31       vyhodnot_reseni();
32       uloz_feromon_na_navstivene_hrany();
33       aktualizuj_smerovaci_tabulu();
34    }
35    zrus_mravence();
36 }
```

Feromon můžeme ukládat buď použitím procedury *aktualizace_feromonu_krok-po-kroku,* kdy se feromon ukládá okamžitě během

pohybu mravence, nebo pomocí procedury *aktualizace_feromonu_zpozdena* až po vygenerování kompletního řešení. Je možno též modifikovat řešení pomocí démonických akcí, které mohou ovlivňovat rozložení feromonu z vnějšku, s využitím globální informace [3].

3.4 APLIKACE

Jako první byl algoritmus využívající chování mravenců aplikován na problém obchodního cestujícího (díky jeho podobnosti s úlohou, kterou řeší reální mravenci). ACO však lze aplikovat na širokou škálu problémů.

Problémy, na které lze ACO aplikovat, lze rozdělit na statické a dynamické.

Díky lokalitě komunikace se ACO nehodí pro problémy s vysokými počty sousedních stavů [2][3].

Tabulka 3.1 Příklad problémů řešitelných pomocí ACO

Statické problémy	Dynamické problémy
Traveling salesman	Connection-oriented network routing
Quadratic assignment	Connection-less network routing
Job-shop sheduling	
Vehicle routing	
Sequential ordering	
Graph coloring	
Shortest common subsequence	
Bin packing & Cutting Stock	

3.5 PŘÍKLAD ŘEŠENÍ POMOCÍ ACO

V této kapitole je uveden názorný příklad řešení problému pomocí ACO. Jako ukázka byl zvolen problém obchodního cestujícího, a to pro jeho názornost.

3.5.1 Problém obchodního cestujícího (TSP)

Na mapě je N měst a je známa vzájemná vzdálenost pro každá dvě města. Úkolem je zjistit takové pořadí návštěvy měst, aby každé město bylo navštíveno

právě jednou, výsledná uražená vzdálenost co nejkratší a cestující se nakonec vrátil do výchozího města.

Při řešení běžným enumerativním algoritmem roste náročnost nalezení optimálního řešení exponenciálně s velikostí problému. Problém TSP byl mnohokrát řešen i jinými metodami a bude tedy možné snadno srovnat výsledky s řešením pomocí ACO.

3.5.2 Řešení TSP pomocí ACO

Níže popsaný algoritmus využívá jak kladné zpětné vazby založené na analogii se značením cesty feromony u reálných mravenců, tak i záporné zpětné vazby (vypařování feromonu). Tím se omezí možnost uváznutí v lokálním extrému.

```
1  //inicializace
2  for(každá hrana(i,j)){
3        τ_{i,j}(0) = τ_0;
4  }
5  for(k = 1 až m)){
6        Umísti mravence k do náhodně vybraného města;
7  }
8  //Nechť T+ je nejkratší od začátku nalezená cesta a L+ její
   délka
9  //hlavní smyčka
10 for(t = 1 až t_max){
11       for(k = 1 až m){
12             //Postavit cestu T^k(t) aplikací násl. kroků n-1
   krát
13             if (existuje alespoň jedno město j∈seznam
   kandidátů){
14                   Vybrat příští město j,j∈J_i^k, mezi cl městy
   na seznamu kandidátů jako:
```

$$j = \begin{cases} \arg\max_{u \in J_i^k} \left\{ \left[\tau_{iu}(t)\right] \cdot \left[\eta_{iu}\right]^{\beta} \right\} & pro\ q \le q_0; \\ J & pro\ q > q_0, \end{cases}$$

```
   , kde J∈J_i^k, je vybráno
```

$$p_{iJ}^{k}(t) = \frac{\left[\tau_{iJ}(t)\right] \cdot \left[\eta_{iJ}\right]^{\beta}}{\sum_{u \in J_i^k} \left[\tau_{iu}(t)\right] \cdot \left[\eta_{iu}\right]^{\beta}},$$

```
   dle pravděpodobnosti ;
15             }else{
16                   vybrat nejbližší j∈J_i^k;
17             }
18       Po každém přesunu mravenec k aplikuje lokální
   aktualizační    pravidlo τ_ij(t)← (1 - ρ).τ_ij(t) + ρ.τ_0;
19       }
20       for(k = 1 až m)){
21             Vypočítat délku L^k(t) cesty T^k(t) produkované
   mravencem k;
```

```
22        }
23        if(byla nalezena lepší cesta){
24              aktualizovat T+ a L+;
25        }
26        for(každá hrana(i,j)){
27              Aktualizovat feromonové stopy aplikací
    následujícího pravidla:  τij(t)← (1 - ρ).τij(t) + ρ.Δ τij(t) kde Δ τij(t) =
    1/L+;
28        }
29        for(každá hrana(i,j)){
30                    τij(t+1) = τij(t);
31        }
32  }
```

Tabulka 3.2 Popis proměnných algoritmu

τ	Hodnota feromonu
m	Počet mravenců
T	Cesta
T+	Nejkratší cesta
q	Náhodná veličina s rovnoměrným rozložením[0,1]
q_0	Nastavitelný parametr z intervalu [0,1]
j	Město
J_i^k	Města dosud nenavštívená mravencem
η	Dohlednost
p	Pravděpodobnost přechodu
L	Délka cesty
L+	Délka nejkratší cesty
ρ	Odpařování feromonu

4. IMPLEMENTACE ALGORITMŮ

4.1 ÚVOD

Implementovány byly tři algoritmy, a to Tabu Search, Simulated Annealing a ACO. Všechny tři algoritmy byli aplikovány na symetrický TSP (ovšem provedená implementace by měla bez problému fungovat i pro asymetrický TSP). Vstupem u všech algoritmů je symetrická matice hran orientovaného grafu, ve kterém vrcholy představují jednotlivá města a hrany cenu přechodu z jednoho města do druhého. Hrany mohou mít velikost od 0 do maximální hodnoty typu double. Hrana s hodnotou 0 je hranou neexistující. V algoritmech samotných jsou neexistující hrany reprezentovány hranou s cenou o několik řádů vyšší než jsou ceny existujících hran.

K implementaci byl použit jazyk C++ v prostředí Microsoft Visual C++ 6.0.

4.2 TABU SEARCH

4.2.1 Kostra algoritmu

```
1   for(i = 0; i < C_PocetIteraci; i++)
2   {//pro všechny iterace
3     V_LokOptCesta->DelkaCesty = V_VelHrany * C_NeexistujiciHrana
    + 1;// lokalní minimum na maximum
4     for(j = 0; j < V_PocetOperaci; j++)
5     {//pro všechny operace
6       V_NovaCesta = Transformuj(V_OpSezn, j, V_Cesta,
    V_NovaCesta);                          //transformuj dle operace
    pokud není v tabu
7       SpoctiCenu(V_Hrany, V_NovaCesta);//přepočítej délku nového
    řešení

8       if ((V_NovaCesta->DelkaCesty < V_LokOptCesta->DelkaCesty)
    && (!JeTabu(j , V_TabuList, C_VelikostTabu)))
9       {//prohoď seznamy měst
10        V_LokOptCesta->DelkaCesty = V_NovaCesta->DelkaCesty;
11        P_Akt = V_LokOptCesta->NMesta;
12        V_LokOptCesta->NMesta = V_NovaCesta->NMesta;
13        V_NovaCesta->NMesta = P_Akt;
14        V_DoTabu = j;
15      }//endif
16    }/endfor operace
17    if (V_LokOptCesta->DelkaCesty < V_VyslCesta->DelkaCesty)
18    {//nová nejlepší cesta
19      V_VyslCesta->DelkaCesty = V_LokOptCesta->DelkaCesty;
```

```
20      P_Akt = V_VyslCesta->NMesta;
21      P_AktN = V_LokOptCesta->NMesta;
22      do
23      {
24        P_Akt->Mesto = P_AktN->Mesto;
25        P_Akt = P_Akt->Dalsi;
26         P_AktN = P_AktN->Dalsi;
27      }while((P_Akt != NULL) && (P_AktN != NULL));
28    }//endif
29    VlozDoTabu(V_DoTabu, V_TabuList, C_VelikostTabu);//vloží
   provedenou operaci do tabu
30    V_Cesta->DelkaCesty = V_LokOptCesta->DelkaCesty;
31    P_Akt = V_Cesta->NMesta;
32    P_AktN = V_LokOptCesta->NMesta;
33    do
34    {
35      P_Akt->Mesto = P_AktN->Mesto;
36      P_Akt = P_Akt->Dalsi;
37      P_AktN = P_AktN->Dalsi;
38    }while((P_Akt != NULL) && (P_AktN != NULL));
39  }//endfor iterace
```

4.2.2 Parametry algoritmu

Celý algoritmus Tabu Search byl implementován jako funkce:

```
1   T_Cesta *TabuSearch(double **V_Hrany, int V_VelHrany, int
    C_NeexistujiciHrana, int C_VelikostTabu, int
    C_ProhazovaciVzdalenost, int C_PocetIteraci)
```

Kde typ T_Cesta je definován jako:

```
1   typedef struct
2   {
3     T_NMesta *NMesta;
4     double DelkaCesty;
5   } T_Cesta;
```

a typ T_NMesta jako:

```
1      typedef struct NMesta
2      {
3        int Mesto;
4        struct NMesta *Dalsi;
5      }T_NMesta;
```

Parametr *V_Hrany* vyžaduje ukazatel na trojrozměrné pole typu double, ve kterém jsou uloženy hodnoty hran (vzdálenosti jednotlivých měst) orientovaného grafu počítaného problému. Parametr *V_VelHrany* udává rozměr pole hran (počet měst).

Ostatní parametry mají význam řídicích parametrů algoritmu a jsou podrobněji popsány v následující tabulce. Řídicími parametry jsou míněny ty, jejichž různým nastavením můžeme pozitivně (a samozřejmě i negativně) ovlivnit chování algoritmu.

Tabulka 4.1 Řídicí parametry Tabu Search

Název	Popis	Defaultní hodnota
C_PocetIteraci	Celkový počet iterací	10
C_NeexistujiciHrana	Hodnota neexistující hrany	10000
C_VelikostTabu	Maximální počet položek v tabulistu	10
C_ProhazovaciVzdalenost	Parametr udávající počet operací při hledání nové cesty. Hodnota 0 znamená maximální počet operací.	10

4.2.3 Popis funkcí algoritmu

Implementovaný algoritmus Tabu Search používá několik funkcí. Jejich stručný popis je uveden v následující tabulce.

Tabulka 4.2 Popis funkcí Tabu Search

Deklarace	Popis
`int **InicializujSeznamOperaci(int C_PocetMest, int *P_PocetOperaci, int C_ProhazovaciVzdalenost)`	Vytvoří a inicializuje Seznam operací.
`T_Cesta *VytvorCestu(int C_PocetMest)`	Vytvoří pole obsahující cestu.
`int VlozDoTabu(int V_Vkladane, int *V_TabuList, int C_VelikostTabu)`	Vloží město do tabulistu.
`int SpoctiCenu(double **V_Hrany, T_Cesta *V_Cesta)`	Spočítá cenu daného řešení.
`T_Cesta *Transformuj(int **V_OpSezn, int V_IndexOperace, T_Cesta *V_Cesta, T_Cesta *V_NovaCesta)`	Vytvoří nové řešení (novou cestu).
`int JeTabu(int V_Overovana , int *V_TabuList, int C_VelikostTabu)`	Zjistí zda se dané město nachází v tabulistu.
`int DealokujCestu(T_Cesta *V_Cesta)`	Pomocná funkce sloužící k dealokaci cesty.
`int NactiHrany(char* input, double **V_Pole, int C_VelikostPole, int C_NeexistujiciHrana)`	Pomocná funkce sloužící k načtení pole hran ze vstupního souboru.
`double **VytvorPole2(char* File, int *P_VelikostPole)`	Pomocná funkce sloužící k alokaci dvojrozměrného pole typu double.
`int Loguj(FILE *fa, char *V_Text, double V_Promena, char C_Double, int C_Odradkuj)`	Pomocná funkce sloužící k zápisu do logovacího souboru.

4.3 SIMULATED ANEALING

4.3.1 Kostra algoritmu

```
1  V_Teplota = C_PocatecniTeplota;
2  for(i = 0; i < C_PocetMest; i++)//inicializace nejlepší cesty
3    V_NejlepsiCesta[i] = V_Cesta[i];
4  V_CenaNejlepsiCesty = SpoctiCenu(V_Hrany, V_NejlepsiCesta,
   C_PocetMest); ++)//inicializace ceny nejlepší cesty
5  while (V_Teplota > C_KoncovaTeplota)
6  {//pro celý rozsah teplot
7      for(j = 0; j < C_PocetIteraci; j++)
```

```
8       {//přes všechny iterace
9         for(k = 0; k < C_PocetOperaci-1; k++)
10        {//pro všechny operace ze seznamu
11          GenerujNoveReseni(V_Hrany, V_OpSezn, k, C_PocetMest,
    V_Cesta, V_NovaCesta, C_PocetOperaci);
12          V_CenaCesty = SpoctiCenu(V_Hrany, V_Cesta,
    C_PocetMest);
13          V_CenaNoveCesty = SpoctiCenu(V_Hrany, V_NovaCesta,
    C_PocetMest);
14          if (V_CenaNoveCesty < V_CenaCesty) V_Pravd = 1;
15          else V_Pravd = exp(-(V_CenaNoveCesty -
    V_CenaCesty)/V_Teplota);//pokud mam lepší cestu, ulož
16          if (((double)rand() / RAND_MAX) < V_Pravd)
17          {// přijmi řešení?
18             PrijmiReseni(&V_Cesta, &V_NovaCesta);
19             if (V_CenaNoveCesty < V_CenaNejlepsiCesty)
20             {//mám nove nejlepší řešení
21                for(i = 0; i < C_PocetMest; i++)
22                     V_NejlepsiCesta[i] = V_Cesta[i];
23                V_CenaNejlepsiCesty = SpoctiCenu(V_Hrany,
    V_NejlepsiCesta, C_PocetMest);
24             } //endif
25           } //endif
26        } //endfor operace
27      }//endfor iterace
28    V_Teplota = V_Teplota * C_Beta;
29 }//endwhile teplota
```

4.3.2 Parametry algoritmu

Algoritmus Simulated Annealing byl implementován jako funkce:

```
1  int *Annealing(double **V_Hrany, int C_PocetMest, double
   C_PocatecniTeplota, double C_KoncovaTeplota, double C_Beta,
   int C_PocetIteraci, int C_ProhazovaciVzdalenost)
```

Parametr *V_Hrany* má stejný význam jako u předchozího algoritmu (jedná se o matici hran) a parametr *C_PocetMest* udává rozměr pole hran (počet měst).

Ostatní parametry mají význam řídicích parametrů algoritmu a jsou podrobněji popsány v následující tabulce.

Tabulka 4.3 Řídicí parametry algoritmu Simulated Annealing

Název	Popis	Defaultní hodnota
C_PocetIteraci	Celkový počet iterací	10
C_ProhazovaciVzdalenost	Parametr udávající počet operací při hledání nové cesty. Hodnota 0 znamená maximální počet operací.	10
C_PocatecniTeplota	Počáteční teplota, při které započínáme žíhání.	3000
C_KoncovaTeplota	Konečná teplota, při které žíhání končí.	1
C_Beta	Parametr udávající rychlost poklesu teploty.	0,8

4.3.3 Popis funkcí algoritmu

Stejně jako předchozí algoritmus, i Simulated Annealing používá několik funkcí. Jejich stručný popis je uveden v následující tabulce.

Tabulka 4.4 Popis funkcí Simulated Annealing

Deklarace	Popis
`int **InicializujSeznamOperaci(int C_PocetMest, int *P_PocetOperaci, int C_ProhazovaciVzdalenost)`	Vytvoří a inicializuje Seznam operací.
`int VytvorPocatecniReseni(double **V_Hrany, int C_PocetMest,int *V_Cesta)`	Vytvoří počáteční řešení (uzavřená cesta přes existující hrany).
`double SpoctiCenu(double **V_Hrany, int *V_Cesta, int C_DelkaCesty)`	Spočítá cenu daného řešení.
`int JeCesta(double **V_Hrany, int V_Z, int V_Do)`	Zjistí existenci cesty.
`int GenerujNoveReseni(double **V_Hrany, int **V_OpSezn, int V_IndexOperace, int C_DelkaCesty, int *V_Cesta, int *V_NovaCesta, int C_PocetOperaci)`	Vytvoří nové řešení (novou cestu).
`int PrijmiReseni(int **P_Cesta, int **P_NovaCesta)`	Přijme řešení.
`int NactiHrany(char* input, double **V_Pole, int C_VelikostPole, int C_NeexistujiciHrana)`	Pomocná funkce sloužící k načtení pole hran ze vstupního souboru.
`double **VytvorPole2(char* File, int *P_VelikostPole)`	Pomocná funkce sloužící k alokaci dvojrozměrného pole typu double.
`int Loguj(FILE *fa, char *V_Text, double V_Promena, char C_Double, int C_Odradkuj)`	Pomocná funkce sloužící k zápisu do logovacího souboru.

4.4 ANT COLONY ALGORITHM

4.4.1 Kostra algoritmu

```
1   for(i = 0; i < C_PocetIteraci; i++)//for všechny iterace
2   {
3     for(j = 0; j < C_PocetMravencu; j++)//for všechny mravence
4     {
a.        V_Mravenec = InicializujMravence(V_VelHrany);
b.        //inicalizace aktualni delky a cesty
c.        V_AktDelkaCesty = 0;
```

```
d.        for(k = 0; k < V_VelHrany; k++) V_AktCesta[k] = -1;
e.        V_AktCesta[0] = V_Mravenec->AktMesto;
5       for(k = 0; k < V_VelHrany-1; k++)//pro vsechna mesta
6       {
7         V_SmerMravence = SmerovaniMravence(V_Hrany,
    V_SmerovaciTabulka, V_VelHrany, C_MaxMest, V_Mravenec, C_Beta,
    C_q0, C_NeexistujiciHrana);
8         PohniMravencem(V_Hrany, V_Mravenec, V_SmerMravence,
    C_Ro, V_Tau0, V_AktCesta);
9       }//endfor města
10      V_AktDelkaCesty = SpoctiDelkuAktCesty(V_AktCesta, V_Hrany,
    V_VelHrany);
11      if(V_MinDelkaCesty > V_AktDelkaCesty)
12      {
13        for(k = 0; k < V_VelHrany; k++)
14        {
15          V_MinCesta[k] = V_AktCesta[k];
16          V_MinDelkaCesty = V_AktDelkaCesty;
17          V_MinIterace = i;
18        }//endfor
19      }//endif
20      ZrusMravence(V_Mravenec);
21    }//end mravenci
22    //ulozeni globalniho feromonu
23    AktualizujGlobalneFeromon(V_Hrany, V_MinCesta, V_VelHrany,
    V_MinDelkaCesty, C_Ro);
24  }//end iterace
```

4.4.2 Parametry algoritmu

Algoritmus ACO byl implementován jako funkce:

```
1   int *AntColony(int V_VelHrany, double ***V_Hrany, double
    V_Tau0, double C_Beta, double C_q0, double C_Ro, int
    C_NeexistujiciHrana, int C_MaxMest, int C_PocetIteraci, int
    C_PocetMravencu, FILE *Pfw_Log, FILE *Pfw_Vyst)
```

Parametr *V_Hrany* má stejný význam jako u předchozího algoritmu (jedná se o matici hran) a parametr *C_VelHrany* udává rozměr pole hran (počet měst). Poslední dva parametry (*Pfw_Log* a *Pfw_Vyst*) jsou ukazatele na otevřený losovací a výstupní soubor.

Ostatní parametry mají opět význam řídicích parametrů algoritmu a jsou podrobněji popsány v následující tabulce.

Tabulka 4.5 Řídicí parametry algoritmu ACO

Název	Popis	Defaultní hodnota
C_PocetIteraci	Celkový počet iterací	100
C_NeexistujiciHrana	Hodnota neexistující hrany	10000
C_PocetMravencu	Počet mravenců	10
C_MaxMest	Počet měst ve směrovací tabulce	5
C_Beta	Omezení dohlednosti	2
C_q0	Omezení výzkumu	0,9
C_Ro	Rozklad feromonu	0,1
V_Tau0	Počáteční hodnota feromonu	1

4.4.3 Popis funkcí algoritmu

V následující tabulce jsou popsány funkce, které jsou součástí algoritmu ACO. Protože je zdrojový kód ACO rozsáhlejší než u předchozích dvou algoritmů, byl tento rozdělen na dvě části.

4.4.3.1 Funkce algoritmu ACO

V následující tabulce jsou popsány funkce, které přímo souvisí s logikou algoritmu ACO.

Tabulka 4.6 Popis funkcí ACO

Deklarace	Popis
`double SpoctiTau0(double ***V_TabHran, int C_VelTabHran, int C_NeexistujiciHrana);`	Vrací hodnotu počátečního feromonu (výpočet heuristikou nejbližšího souseda)
`T_Mravenec *InicializujMravence(int C_PocetMest);`	Vytvoří nového mravence
`int **VytvorSmerovaciTabulku(double ***V_TabHran, int **V_SmerTab, int C_VelTabHran, int C_MaxMest);`	Vytvoří směrovací tabulku
`int SmerovaniMravence(double ***V_TabHran, int **V_SmerTab, int C_VelTabHran, int C_MaxMest, T_Mravenec *V_Mravenec, double C_Beta, double C_q0, int C_NeexistujiciHrana);`	Rozhodne do kterého města mravenec půjde
`int PohniMravencem(double ***V_TabHran, T_Mravenec *V_Mravenec, int C_CiloveMesto, double C_Ro, double C_Tau0, int *V_AktCesta);`	Přesune mravence do z aktuálního do cílového města
`int SpoctiDelkuAktCesty(int *V_AktCesta, double ***V_TabHran, int C_VelTabHran);`	Vypočítá délku aktuálního řešení(cesty)
`int AktualizujGlobalneFeromon(double ***V_TabHran, int *V_MinCesta, int C_VelTabHran, int C_MinDelkaCesty, double C_Ro);`	Globálně aktualizuje hodnoty feromonu na nejlepší cestě

4.4.3.2 Pomocné funkce

V následující tabulce jsou popsány funkce, které nejsou přímo součástí algoritmu ACO, ale slouží k větší přehlednosti programu, logovaní atp.

Tabulka 4.7 Pomocné funkce ACO

Deklarace	Popis
`int *VytvorPole1(int C_Rozmer);`	Alokuje jednorozměrné pole itegerů. Vrací ukazatel na pole.
`int **VytvorPole2(int C_Rozmer1, int C_Rozmer2);`	Alokuje dvojrozměrné pole integerů. Vrací ukazatel na pole.
`double ***VytvorPole3(char* File, int *P_VelikostPole);`	Alokuje trojrozměrné pole hran. Vrací ukazatel na pole. Při neúspěchu vrací 0.
`int NactiHrany(char* input, double ***V_Pole, int C_VelikostPole, int C_NeexistujiciHrana, double C_PocatecniFeromon);`	Naplní pole hran hodnotami ze vstupního souboru. Při neúspěchu vrací 0. Jinak vrací 1.
`int NactiRadek(FILE *input, char *V_Radek, int* P_PocetZnakuRadku)`	Načte jeden řádek vtupního souboru. Pokud již nelze načíst další řádku ze souboru vrací 0. Jinak vrací 1.
`int DealokujHrany(double ***V_Pole, int C_VelikostPole);`	Dealokuje pole hran.
`int ZrusMravence(T_Mravenec *P_Mravenec);`	Dealokuje mravence.
`int Dealokuj2Pole(int **V_Pole,int C_Rozmer);`	Dealokuje dvojrozměrné pole.
`int QuickSort(int **V_Pole, int V_L, int V_P);`	Seřadí jednorozměrné pole. Používá metodu QuickSort.
`int Loguj(char* File, char *V_Text, double V_Promena, char C_Double, int C_Odradkuj);`	Slouží k logování údajů do souboru.

5. SROVNÁNÍ ALGORITMŮ

5.1 ÚVOD

Pro srovnání algoritmů byly provedeny dva druhy testů. Jeden druh sloužil k určení časové náročnosti algoritmu a druhý na učení kvality řešení (nalezení nejkratší cesty).

Parametry počítačové sestavy, na které byly testy prováděny, jsou zapsány v následující tabulce.

Tabulka 5.1 Parametry testovací sestavy

CPU	Core2Duo E6420 2,13GHz
Paměť	2GB DDR2 800MHz CL 5-5-5
Operační Systém	Microsoft Windows XP Professional 32

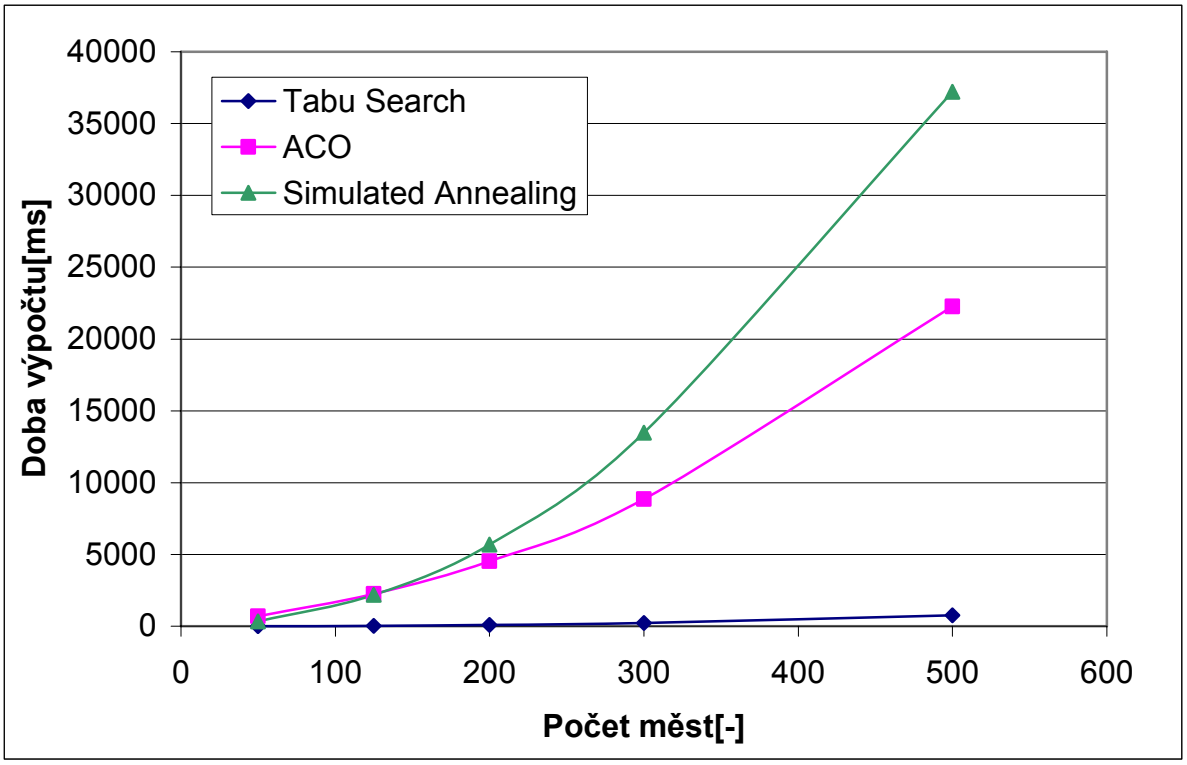
5.2 ČASOVÁ NÁROČNOST ALGORITMŮ

Pro testování časové náročnosti jednotlivých algoritmů bylo použito dat použitých v rámci sdíleného výpočetního projektu TSP [12]. Základní soubor dat obsahoval reálná data o pozicích 512 měst. Pro účely testování byl tento soubor rozdělen na soubory s počtem měst 50, 125, 200, 300, 500.

U algoritmů byly testovány dvě základní vlastnosti. První byl vliv počtu prohledávaných měst na dobu trvání výpočtu a druhou potom byl vliv nastavení některých řídicích parametrů na tuto dobu.. Každé měření bylo provedeno pětkrát a konečný výsledek byl získán jako průměr naměřených hodnot. Dosažené výsledky a použité parametry algoritmů jsou vidět v následující tabulkách a grafech.

Tabulka 5.2 Vliv počtu měst na časové náročnosti algoritmů

	Trvání výpočtu[ms]		
Počet měst	**ACO**	**Tabu Search**	**Simulated Annealing**
50	684	9	344
125	2247	31	2181
200	4537	88	5678
300	8853	234	13478
500	22266	750	37222
Parametry algoritmu	Počet iterací: 100	Počet iterací: 10	Počet iterací: 10
	Cena neexistující hrany: 10000	Cena neexistující hrany: 10000	Maximální vzdálenost prohazovaných prvků: 10
	Počet měst ve směrovací tabulce:15	Maximální vzdálenost prohazovaných prvků: 10	Počáteční teplota: 3000
	Omezení dohlednosti (β): 2	Velikost tabulistu: 10	Koncová teplota: 1
	Omezení výzkumu (q_0):0.9		β: 0,8
	Rozklad feromonu (ρ_0): 0.1		



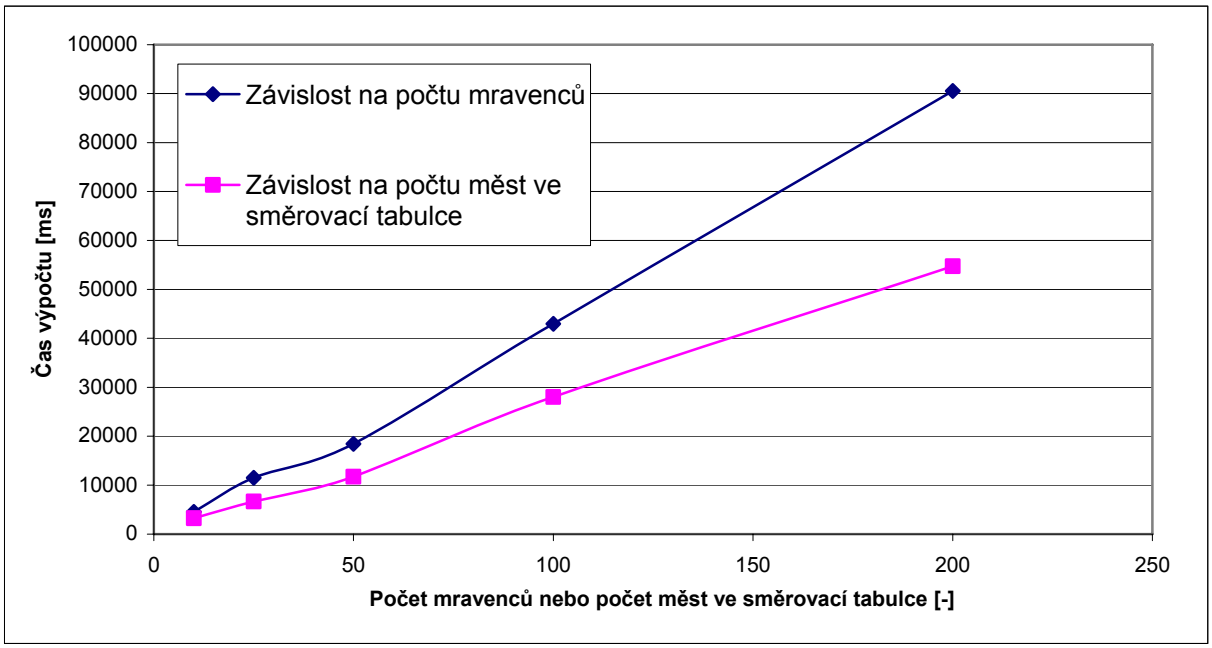
Graf 5.1 Porovnání časové náročnosti algoritmů

Z grafu je dobře patrné, že co se týká časové náročnosti, je na tom nejlépe Tabu Search. Na rozdíl od obou zbývajících algoritmů u něj roste složitost lineárně, což je velmi dobrá vlastnost.

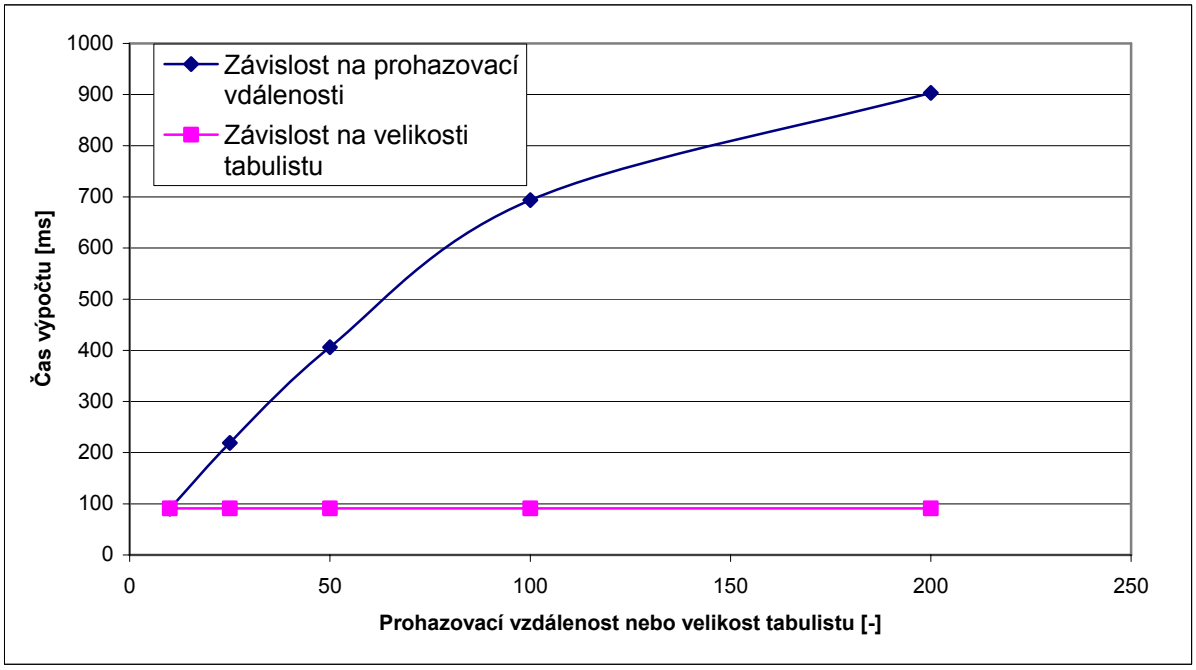
Nyní následuje porovnání vlivu řídicích parametrů na časovou náročnost všech tří algoritmů. Testovány byly pouze parametry, které mohly mít nějaký vliv na časovou náročnost, tzn. nebyly testovány parametry, jež jsou pouze konstantami nějakého z výpočetních vzorců.

Tabulka 5.3 Vliv parametrů na časovou náročnost algoritmů

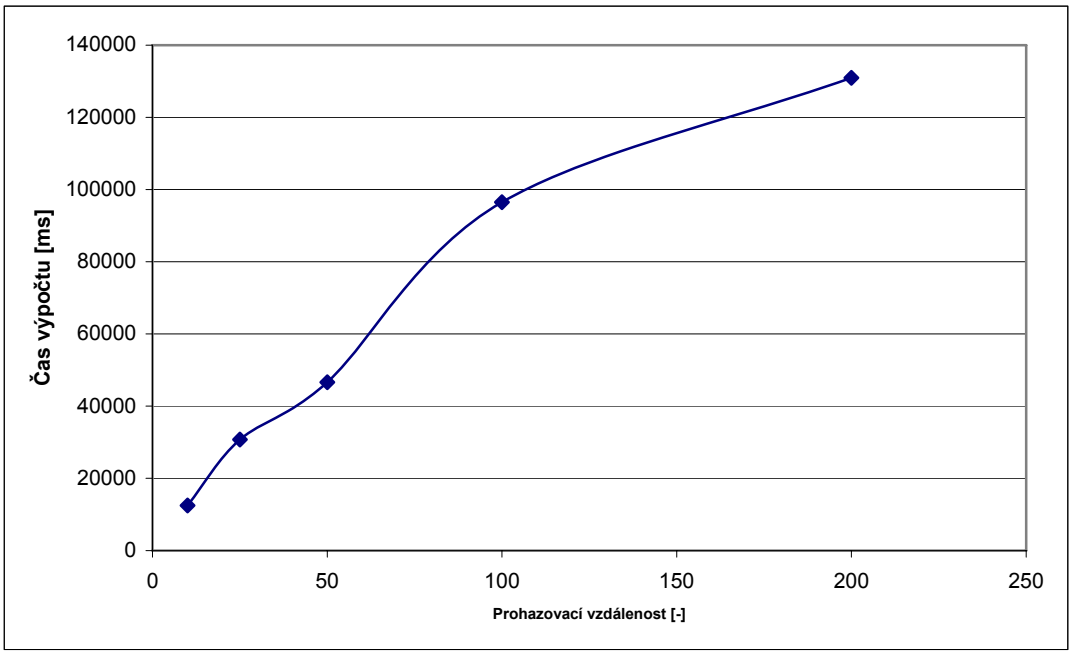
ACO		Tabu Search		Simulated Annealing	
Počet mravenců [-]	**Čas výpočtu [ms]**	**Vzdálenost prohazovaných prvků [-]**	**Čas výpočtu [ms]**	**Vzdálenost prohazovaných prvků [-]**	**Čas výpočtu [ms]**
10	4505	10	91	10	12472
25	11484	25	219	25	30759
50	18450	50	406	50	46591
100	42956	100	694	100	96506
200	90537	200	903	200	130950
Počet měst ve směrovací tabulce [-]		**Velikost tabulistu [-]**		β [-]	
10	3237	10	91	0,95	56831
25	6650	25	91	0,9	32767
50	11737	50	91	0,8	12706
100	28013	100	91	0,6	5878
200	54764	200	91	0,4	3313
				0,2	1834
				0,1	1478
				0,05	1072



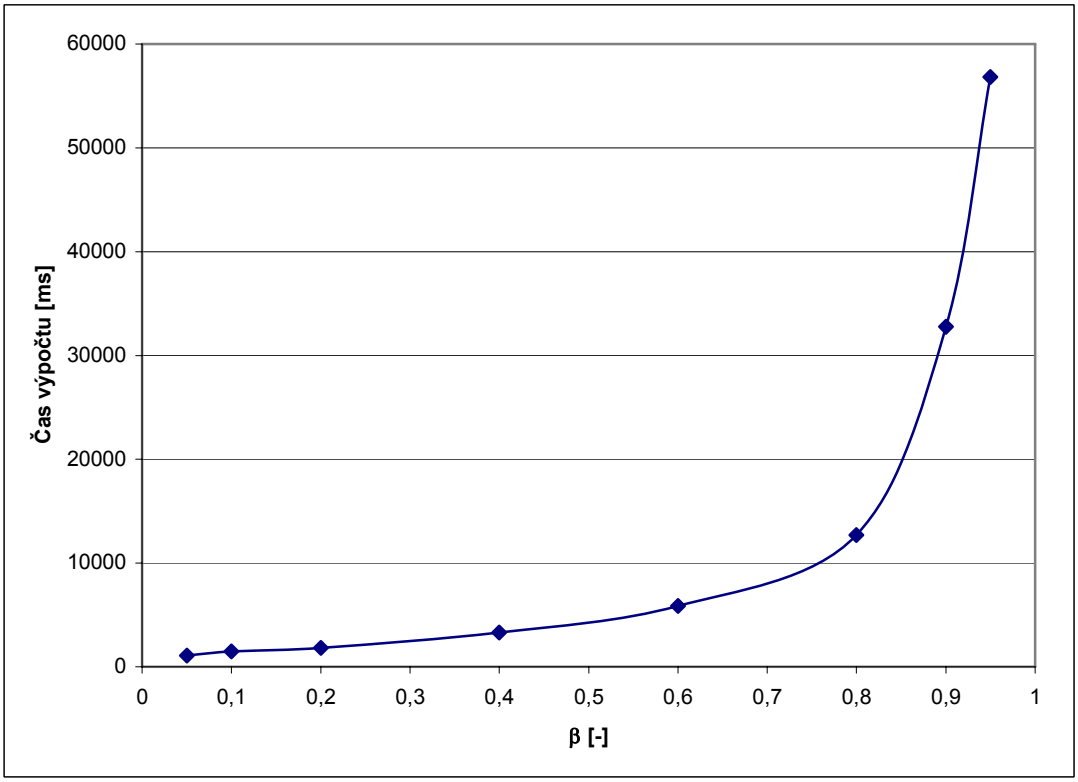
Graf 5.2 Vliv parametrů na časovou náročnost ACO



Graf 5.3 Vliv parametrů na časovou náročnost Tabu Search



Graf 5.4 Vliv prohazovaní vzdálenosti na časovou náročnost Simulated Annealing



Graf 5.5 Vliv parametru β na časovou náročnost Simulated Annealingu

Z předchozích grafů jsou patrné následující skutečnosti.

Časová náročnost algoritmu ACO lineárně roste v závislosti na rostoucím počtu mravenců, kteří prohledávají prostor (viz Graf 5.2). Podobně se algoritmus chová i při zvyšování počtu měst ve směrovací tabulce, avšak nejedná se o tak podstatné zjištění jako je údaj o vlivu mravenců. To proto, že v reálných případech se nikdy nenastavuje počet měst ve směrovací tabulce tak vysoko (viz kapitola 6).

U Tabu Search bylo zjištěno, že časová náročnost výpočtu logaritmicky narůstá s rostoucím prohazovaním vzdáleností prvků při generování nového řešení. Naopak velikost tabulistu nemá na časovou náročnost algoritmu vliv (viz Graf 5.3).

Stejně jako u Tabu Search se parametr udávající prohazovaní vzdáleností prvků chová i u Simulated Annealingu (viz Graf 5.4). Vzhledem k tomu, že u obou algoritmů má parametr nejen stejný název, ale i význam a velmi podobnou implementaci, není tento výsledek nijak překvapující. S rostoucím parametrem β roste složitost výpočtu exponenciálně (viz Graf 5.5).

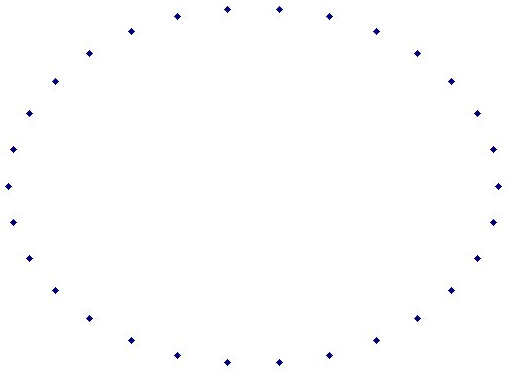
5.3 KVALITA VÝSTUPU

Kvalitou výstupu je míněna míra toho, jak dobře (kvalitně) dokáže daný algoritmus vyřešit úlohu. Konkrétně pro problém TSP je to cesta procházející všemi městy. Čím je cesta kratší (má menší cenu), tím je výsledek lepší.

Pro porovnání kvality řešení poskytovanými jednotlivými algoritmy byly provedeny testy s dvěma soubory dat. Hlavním účelem obou následujících testů bylo zjistit, jak složité je nastavit parametry algoritmů tak, aby byly schopny najít nejlepší cestu mezi městy, a zda vůbec jsou schopny tuto cestu najít.

5.3.1 Test s kruhovým uspořádáním měst

První soubor dat byl vygenerován jako skupina 30 měst uspořádaných do kruhu (viz Obrázek 2.1) a bylo testováno, zda je algoritmus schopen nalézt nejkratší cestu a jak obtížné bylo nastavit řídicí parametry algoritmu, aby tohoto dosáhl. Vzhledem k prostorovému uspořádání měst byla nejkratší cesta předem známa.

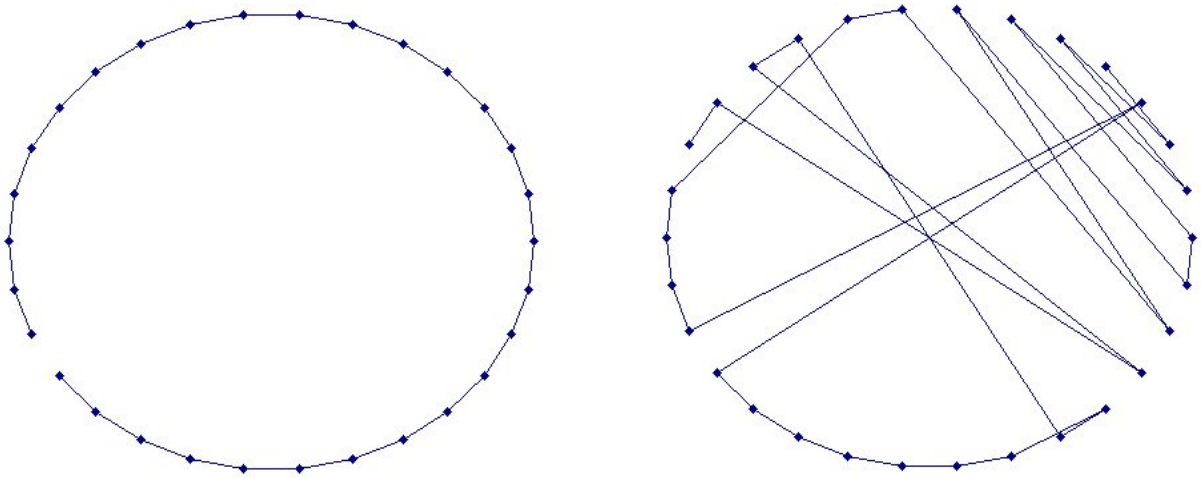


Obrázek 5.1 Prostorové uspořádání měst v prvním testu

Každý z algoritmů byl nejprve spuštěn s defaultním nastavením řídicích parametrů a poté byly tyto parametry (viz Tabulka 2.1) měněny ve snaze o nalezení nejlepší cesty algoritmem. Výsledky je možno vidět na následujících obrázcích.

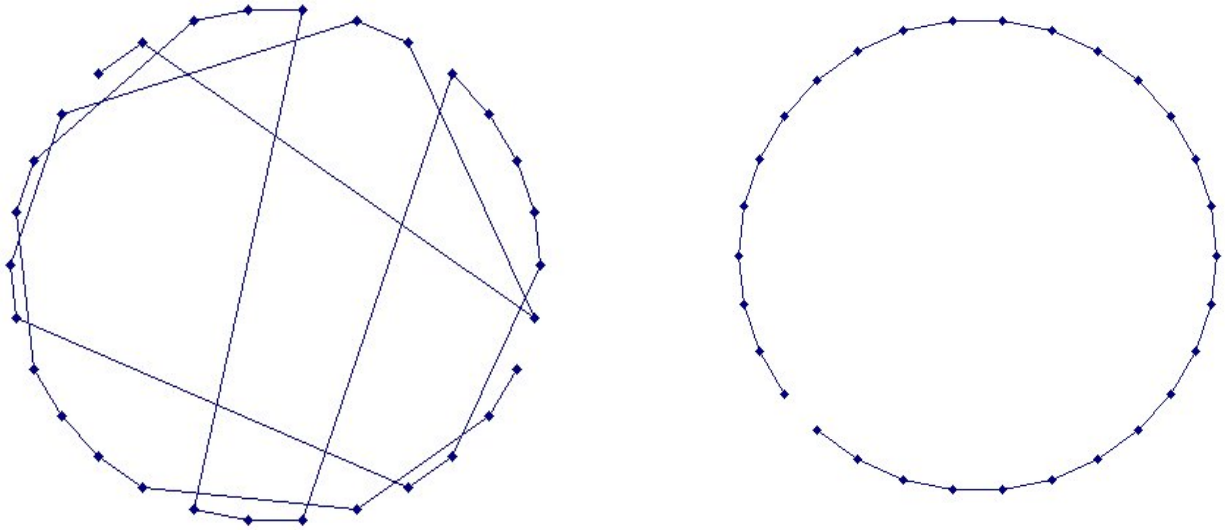
Tabulka 5.4 Hodnoty řídicích parametrů algoritmů

ACO			Tabu Search			Simulated Annealing		
Parametr	**Defaultní hodnota**	**Hodnota při nejlepším dosaženém výsledku**	**Parametr**	**Defaultní hodnota**	**Hodnota při nejlepším dosaženém výsledku**	**Parametr**	**Defaultní hodnota**	**Hodnota při nejlepším dosaženém výsledku**
Počet iterací	50		Počet iterací	1000	1000	Počet iterací	10	10
Počet měst ve směrovací tabulce	5		Maximální vzdálenost prohazovaných prvků	10	0	Maximální vzdálenost prohazovaných prvků	5	0
Počet mravenců	10		Cena neexistující hrany	1000	1000	Počáteční teplota	2000	4500
Cena neexistující hrany	10000		Velikost tabulistu	10	10	Koncová teplota	1	0,001
Omezení dohlednosti (β)	2					β	0,95	0,95
Omezení výzkumu (q_0)	0,9		Hodnota 0 u maximální vzdálenosti prohazovaných prvků má význam maximální vzdálenosti.					
Rozklad feromonu (ρ_0)	0,1							

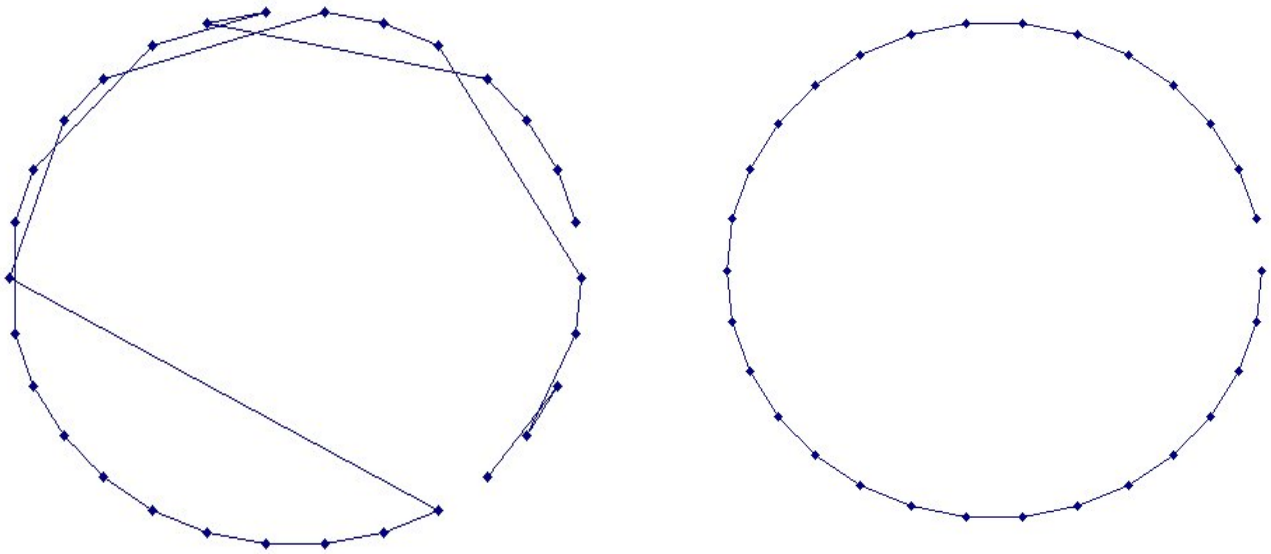


Obrázek 5.2 Cesta kruhem ACO

(vlevo – default, vpravo – při nevhodném nastavení parametrů)



Obrázek 5.3 Cesta kruhem Tabu Search (vlevo – default, vpravo – nejlepší)



Obrázek 5.4 Cesta kruhem Simulated Annealing

(vlevo – default, vpravo – nejlepší)

Z předcházejících obrázků a ze skutečností pozorovaných v průběhu testování vyplývají následující fakta.

Algoritmus ACO nalezl nejlepší cestu již při prvním spuštění, což je velmi dobrý výsledek. Proto bylo zkoušeno více různých nastavení parametrů algoritmu a bylo zjištěno, že jejich nevhodným nastavením (zejména nevhodně nastaveným počtem měst ve směrovací tabulce) lze docílit nenalezení optimální cesty. Nicméně se algoritmus ACO ukázal na daném testovacím souboru velmi necitlivým na zadané parametry a téměř vždy byl schopen nalézt nejlepší cestu.

Na rozdíl od ACO se algoritmu Tabu Search nepodařilo při defaultním nastavení parametrů nalézt nejkratší cestu. Avšak k jejímu nalezení postačila drobná úprava parametrů. Je třeba poznamenat, že algoritmus Tabu Search podával dosti se různící výsledky i při konstantním nastavení parametrů. Dalo by se z toho usuzovat, že je značně citlivý na výchozí město, tj. město, od kterého začíná algoritmus prohledávat prostor.

U algoritmu Simulated Annealing bylo zjištěno, že tento nebyl schopen při defaultním nastavení parametrů najít nejlepší cestu. K jejímu nalezení bylo potřeba

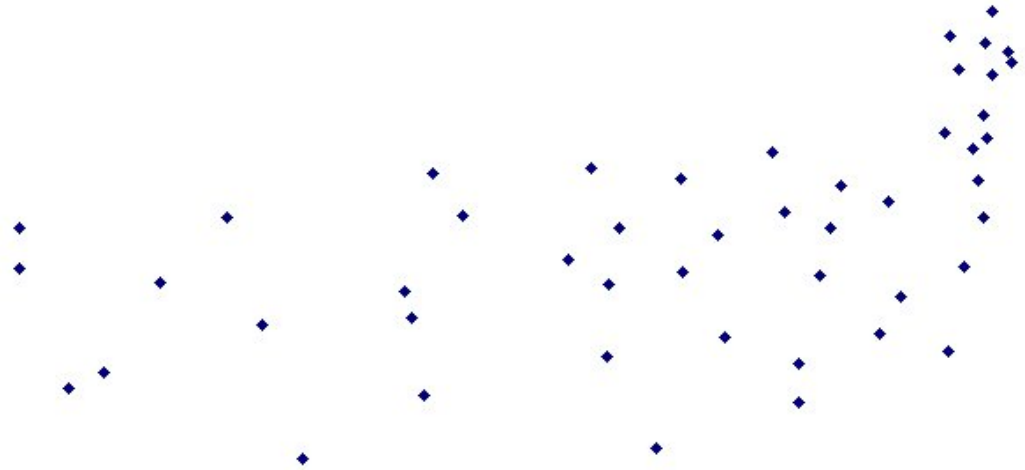
vyzkoušet značné množství možných kombinací parametrů. Subjektivně by se dalo tvrdit, že tento algoritmus prokázal v kruhovém testu nejhorší výsledky.

5.3.2 Test na reálných datech

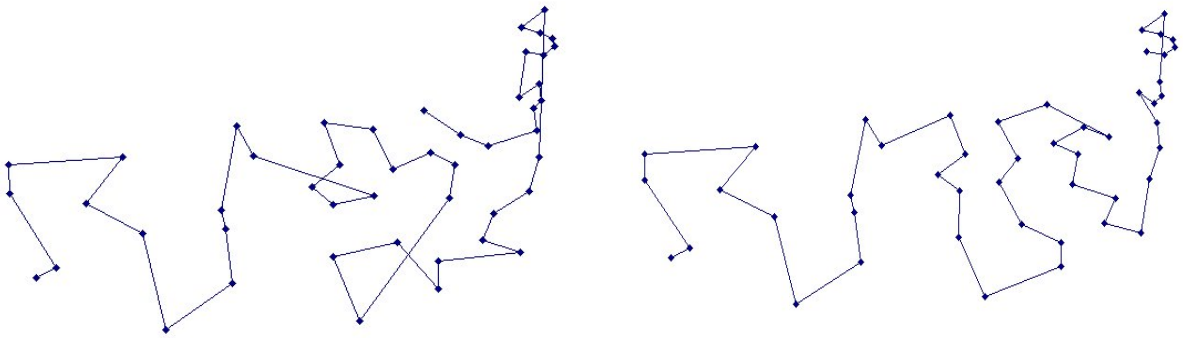
Pro druhý test kvality řešení algoritmů byl použit soubor dat obsahující 48 hlavních měst USA (viz Obrázek 2.1) [12]. Test je podobný předchozímu s tím rozdílem, že byl algoritmy prohledáván prostor založený na reálných datech, a tudíž nebylo předem známo optimální řešení problému. Grafické znázornění pozic měst a nastavení řídicích parametrů je vidět v následující tabulce a obrázku.

Tabulka 5.5 Hodnoty řídicích parametrů algoritmů

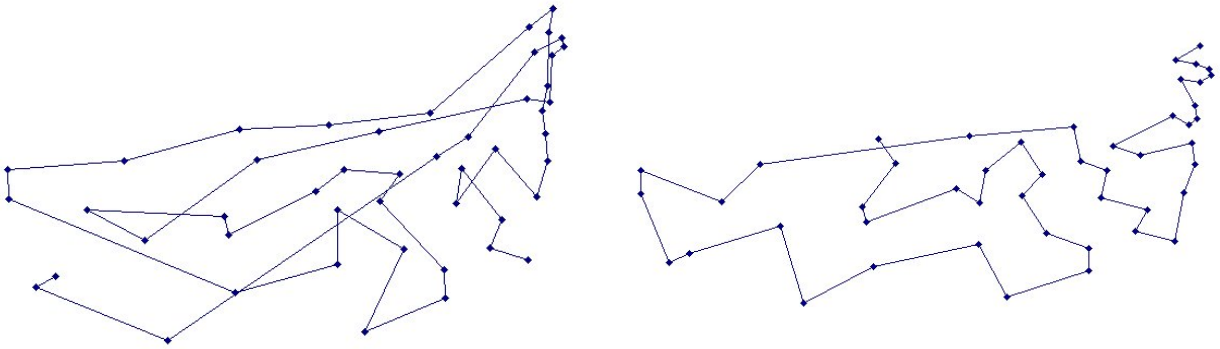
ACO			Tabu Search			Simulated Annealing		
Parametr	**Defaultní hodnota**	**Hodnota při nejlepším dosaženém výsledku**	**Parametr**	**Defaultní hodnota**	**Hodnota při nejlepším dosaženém výsledku**	**Parametr**	**Defaultní hodnota**	**Hodnota při nejlepším dosaženém výsledku**
Počet iterací	50	1000	Počet iterací	1000	1000	Počet iterací	10	10
Počet měst ve směrovací tabulce	15	4	Maximální vzdálenost prohazovaných prvků	10	0	Maximální vzdálenost prohazovaných prvků	5	0
Počet mravenců	10	25	Cena neexistující hrany	1000	1000	Počáteční teplota	2000	3000
Cena neexistující hrany	10000	10000	Velikost tabulistu	5	20	Koncová teplota	1	1
Omezení dohlednosti (β)	2	2000				β	0,95	0,95
Omezení výzkumu (q_0)	0,9	0,9	Hodnota 0 u maximální vzdálenosti prohazovaných prvků má význam maximální vzdálenosti.					
Rozklad feromonu (ρ_0)	0,1	0,1						



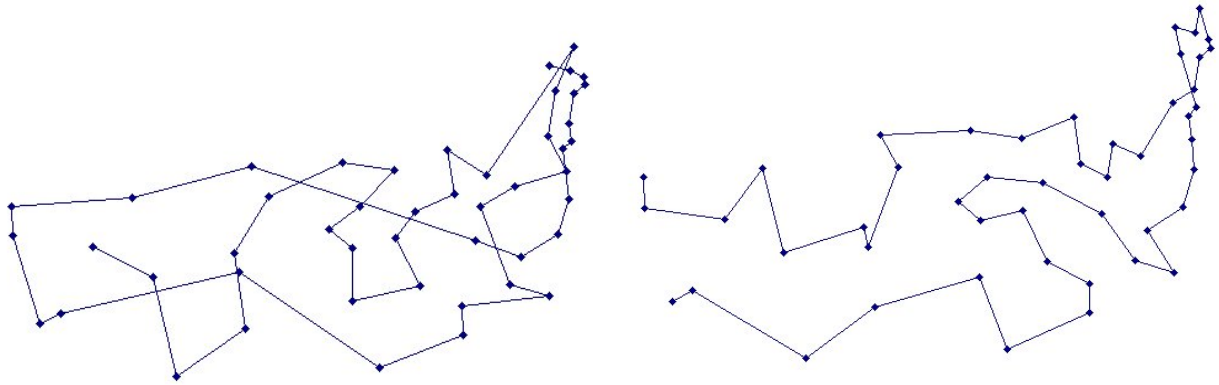
Obrázek 5.5 Prostorové uspořádání měst v druhém testu



Obrázek 5.6 Cesta 48 městy ACO (vlevo – default, vpravo – nejlepší)



Obrázek 5.7 Cesta 8 městy Tabu Search (vlevo – default, vpravo – nejlepší)



Obrázek 5.8 Cesta 48 městy Simulated Annealing

(vlevo – default, vpravo – nejlepší)

Pro úplnost jsou ještě v následující tabulce uvedeny dosažené délky cest.

Tabulka 5.6 Délky nalezených nejlepších cest 48 městy

Algoritmus	Cena nalezené nejmenší cesty při defaultním nastavení parametrů	Cena nejlepší nalezené cesty
ACO	35458,94592	32200,20674
Tabu Search	43972,74938	35277,21995
Simulated Annealing	43007,13655	32842,99519

Z uvedených výsledků testu je jasně vidět, že nejlépe ze všech tří algoritmů dopadl ACO. Ten nejenže nalezl optimální cestu, ale dokázal i poskytnout relativně dobré řešení i při defaultním nastavení parametrů. Druhý nejlepší výsledek podal Simulated Annealing. Nejhůře dopadl algoritmus Tabu Search. Nalezl nejhorší řešení a i to bylo vcelku obtížné získat. Tím je míněna skutečnost, že při vícenásobném spuštění algoritmu byla výstupem velká škála možných řešení s vysokým rozptylem hodnot.

Výsledky by se daly shrnou i tak, že algoritmy poskytovaly tak kvalitní řešení, jak moc jsou samy kvalitní (složité).

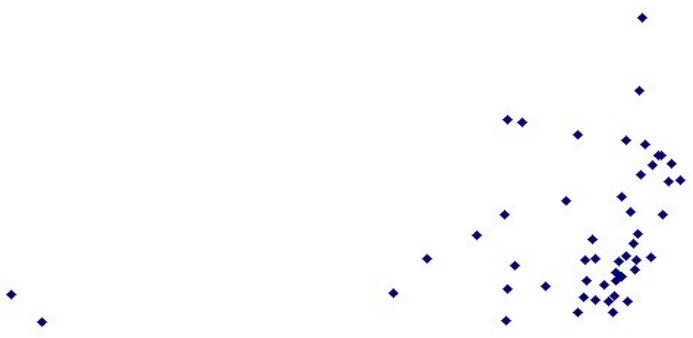
6. VLIV PARAMETRŮ ACO

Předchozí kapitola byla věnována porovnání časové náročnosti zpracování a kvality výstupu poskytovaném jednotlivými algoritmy. Nyní se zaměříme na algoritmus ACO a pokusíme se zjistit, jak řídicí parametry ovlivňují kvalitu nalezeného řešení. Tato znalost je důležitá pro jejich správné nastavení a tím dosažení co nejlepšího výsledku.

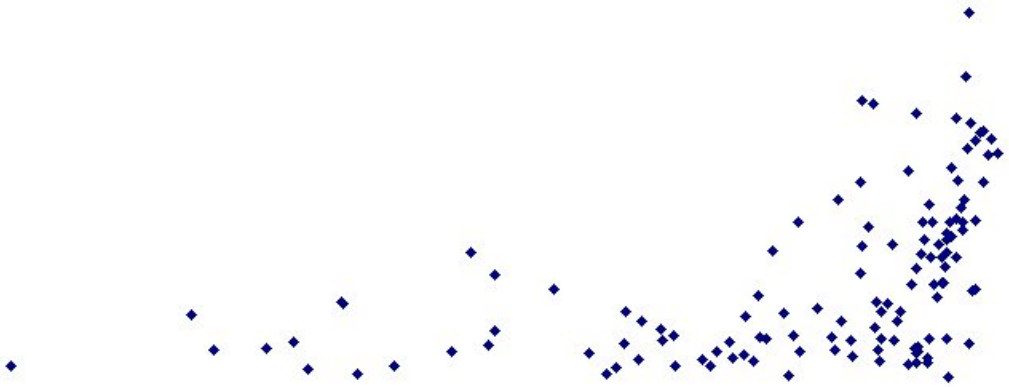
Algoritmus ACO byl testován na množině 125 a 50 měst (jedná se o města v USA). Grafické znázornění množiny měst je možno vidět na následujících obrázcích. Test byl prováděn tak, že algoritmus ACO byl postupně aplikován na zadaný problém (projití 125 či 50 městy), přičemž pokaždé byl jeden zkoumaný parametr měněn a ostatní zůstávaly konstantní (viz Tabulka 6.1). Pro každou hodnotu parametru byl algoritmus spuštěn 10krát a výsledná sledovaná délka cesty byla získána jako průměr těchto deseti hodnot.

Tabulka 6.1 Nastavení řídicích parametrů ACO

Parametr	Hodnota parametru
Počet iterací	1000
Počet měst ve směrovací tabulce	15
Počet mravenců	10
Cena neexistující hrany	10000
Omezení dohlednosti (β)	2
Omezení výzkumu (q_0)	0,9
Rozklad feromonu (ρ_0)	0,1



Obrázek 6.1 Testovací množina 50 měst



Obrázek 6.2 Testovací množina 125 měst

Je důležité upozornit na to, že získané výsledky mají vypovídající hodnotu vztahující se především k datům, na kterých byly získány. Je tu možnost jisté odlišnosti v chování algoritmu při jiných vstupních datech. To se týká hlavně parametrů, jejichž přímý vliv na kvalitu výstupu nebyl prokázán.

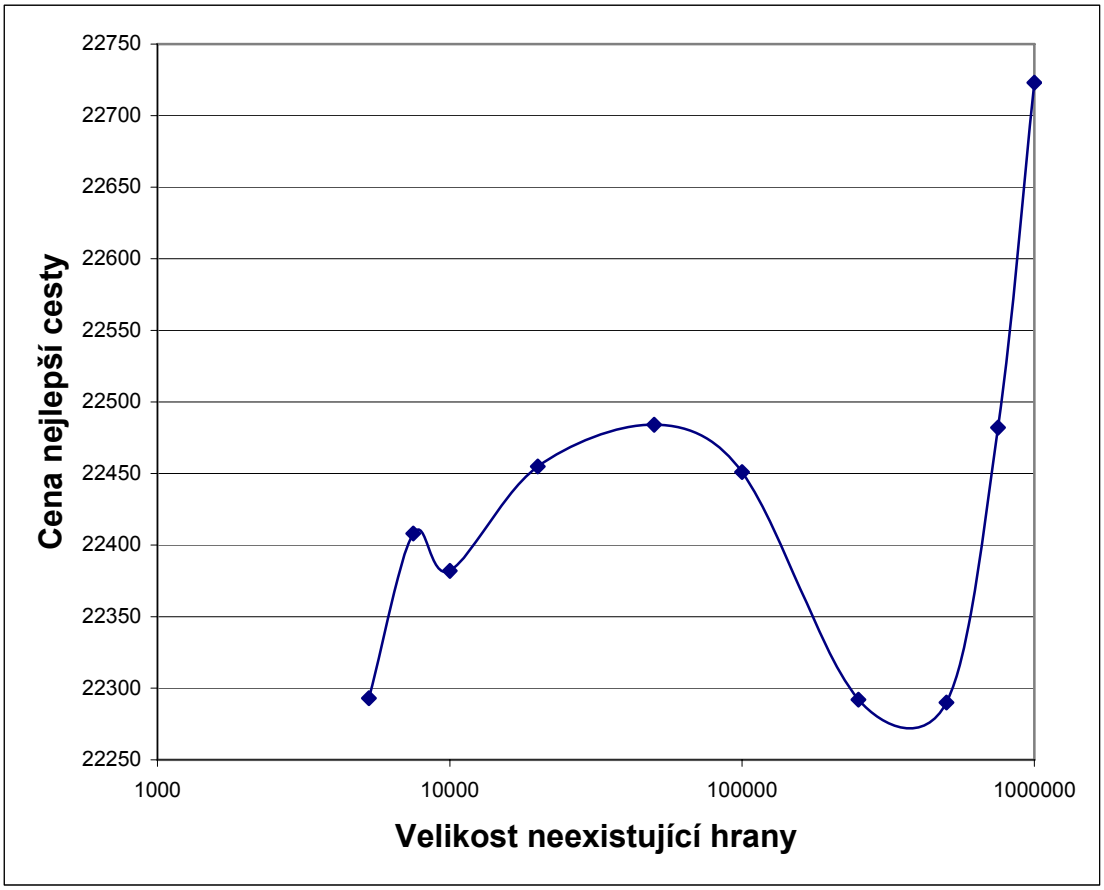
6.1 VELIKOSTI NEEXISTUJÍCÍ HRANY

V orientovaném grafu problému TSP některé hrany mezi městy neexistují, a tudíž přes ně nelze procházet. Z hlediska implementovaného algoritmu to však není tak úplně pravda. Idea je taková, že neexistující hranou procházet lze, avšak její hodnota je natolik vysoká, že se ve výsledné nalezené cestě nebude vyskytovat.

Pro testování byl použita matice hran problému se 125 městy, ve které bylo 20% hran mimo hlavní diagonálu změněno na neexistující. Nejvyšší hodnota hrany v testovacím souboru měla hodnotu 5287.

Tabulka 6.2 Závislost kvality řešení na velikosti neexistující hrany

Velikost neexistující hrany	Cena nejlepší cesty
5287	22293
7500	22408
10000	22382
20000	22455
50000	22484
100000	22451
250000	22292
500000	22290
750000	22482
1000000	22723



Graf 6.1 Závislost kvality výstupu na velikosti neexistující hrany

Z grafu je vidět, že nejlepších výsledků bylo dosaženo s nastavenými hodnotami neexistujících hran přibližně mezi 250000 a 50000. To je přibližně padesát až stonásobek největší hrany vyskytující se v testovacím souboru a to je

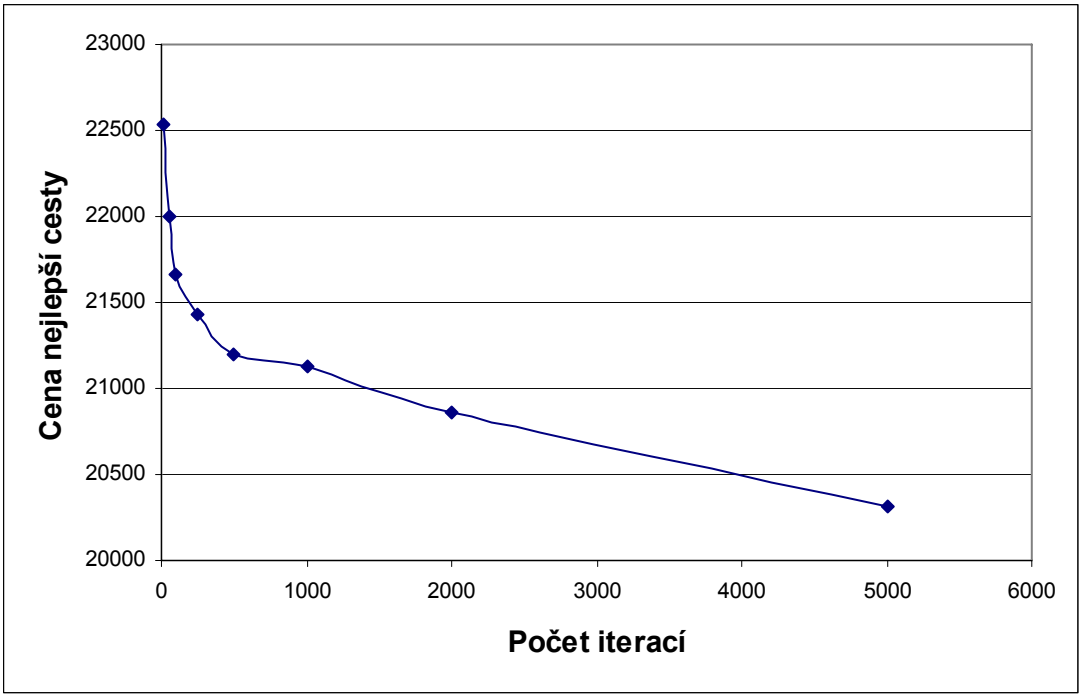
postačující pro správnou funkci algoritmu. Velikost neexistující hrany se nesmí volit příliš nízká, neboť by mohlo dojít k nesprávnému řešení (cesta přes neexistující hranu), ale, jak je vidět z grafu, ani příliš vysoko. Příliš vysoká nastavená hodnota neexistující hrany má za následek zhoršení kvality nalezeného řešení.

6.2 POČET ITERACÍ

Parametr počet iterací udává počet průchodů celým grafem problému, které každý mravenec vykoná. Lze předpokládat, že s počtem iterací se bude kvalita řešení zlepšovat.

Tabulka 6.3 Závislost kvality řešení na počtu iterací

Počet iterací	Cena nejlepší cesty
10	22534
50	21997
100	21662
250	21427
500	21197
1000	21131
2000	20863
5000	20319



Graf 6.2 Závislost kvality řešení na počtu iterací

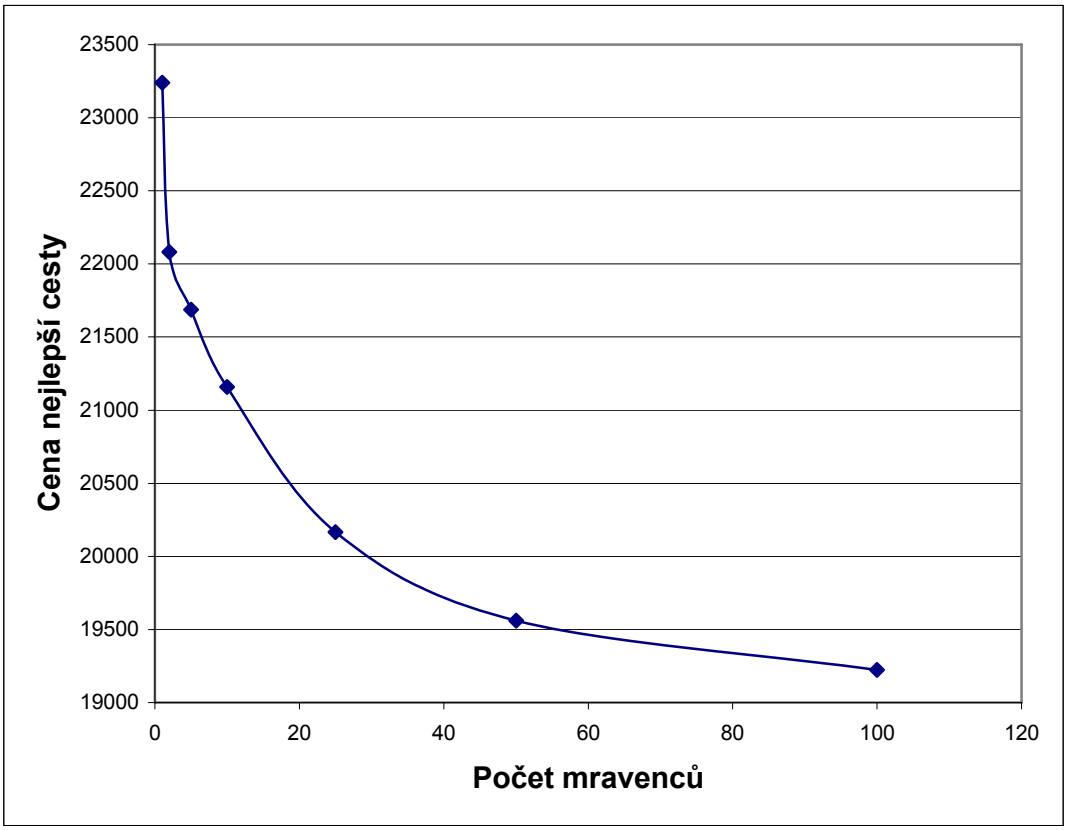
Jak je dobře vidět z grafu, potvrdil se předchozí předpoklad. Se vzrůstajícím počtem iterací roste i kvalita řešení poskytovaného algoritmem.

6.3 POČET MRAVENCŮ

Počet mravenců udává, kolik mravenců se v každé iteraci zúčastní prohledávání grafu problému.

Tabulka 6.4 Závislost kvality řešení na počtu mravenců

Počet mravenců	Cena nejlepší cesty
1	23239
2	22082
5	21688
10	21159
25	20166
50	19562
100	19224



Graf 6.3 Závislost kvality řešení na počtu mravenců

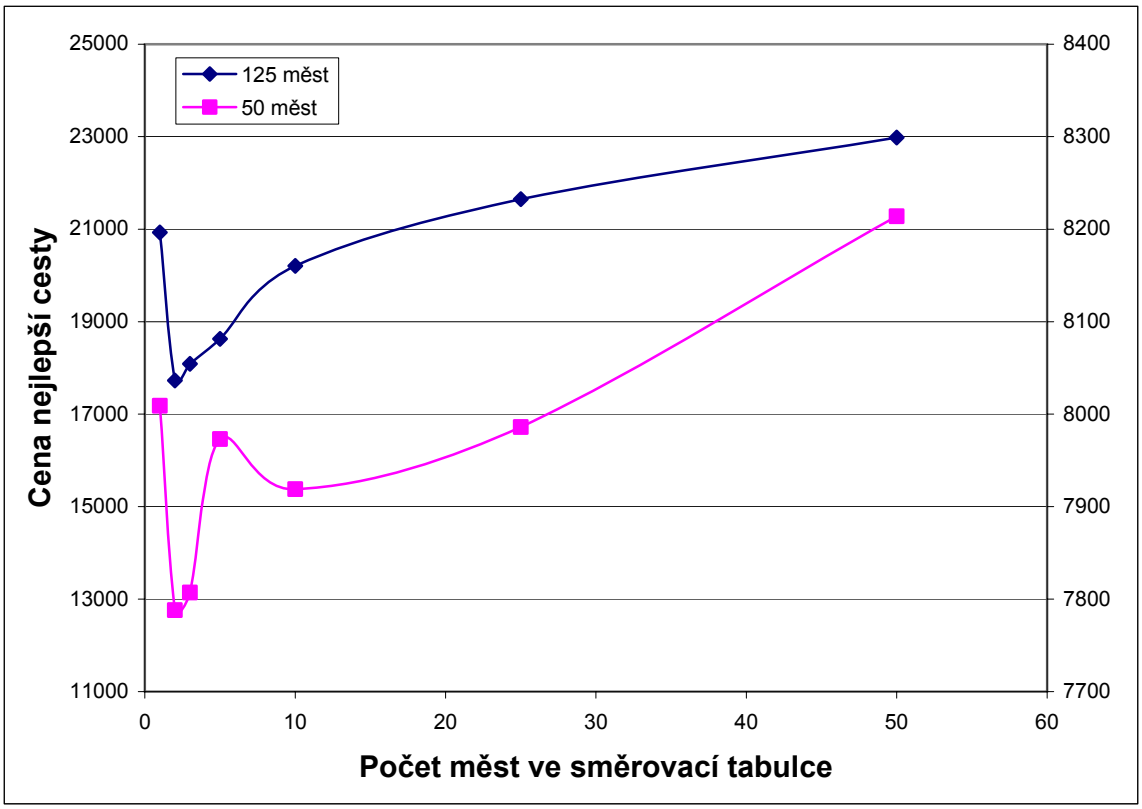
Je dobře vidět, že se vzrůstajícím počtem mravenců roste i kvalita nalezeného řešení.

6.4 POČET MĚST VE SMĚROVACÍ TABULCE

Počet měst ve směrovací tabulce udává počet měst, u kterých se bude mravenec řídit hladinou feromonu při přechodu do dalšího města (detailně viz kapitola 3.5.2). Test závislosti kvality řešení na parametru byl prováděn pro 125 i pro 50 měst.

Tabulka 6.5 Závislost kvality řešení na počtu měst ve směrovací tabulce

	Cena nejlepší cesty	
Počet měst ve směrovací tabulce	125 měst	50 měst
1	20928	8009
2	17728	7788
3	18090	7807
5	18628	7973
10	20205	7919
25	21649	7986
50	22979	8214



Graf 6.4 Závislost kvality řešení na počtu měst ve směrovací tabulce

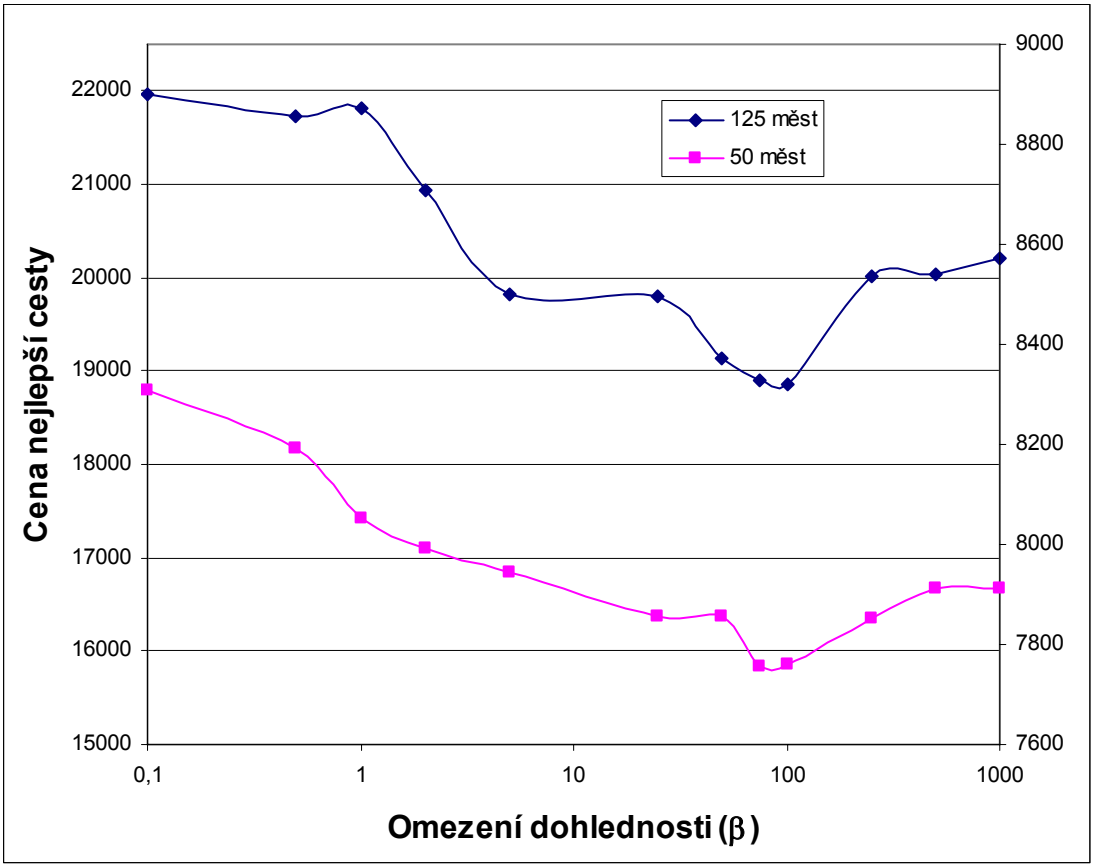
Bylo zjištěno, že nejlepších výsledků dosáhl algoritmus s počtem měst ve směrovací tabulce mezi dva a pět. Z grafu je též možno dedukovat, že se vzrůstajícím počtem měst lze nastavovat i větší počet měst ve směrovací tabulce. Hrozí zde totiž menší nebezpečí radikálního zhoršení nalezeného řešení než u problému s menším počtem měst.

6.5 OMEZENÍ DOHLEDNOSTI

Omezení dohlednosti značený jako β, je parametr regulující relativní váhu dohlednosti. Testování bylo prováděno pro 125 i 50 měst.

Tabulka 6.6 Závislost kvality řešení na omezení dohlednosti

	Cena nejlepší cesty	
Omezení dohlednosti (β)	125 měst	50 měst
0,1	21968	8308
0,5	21720	8192
1	21815	8053
2	20929	7992
5	19825	7945
25	19801	7857
50	19136	7858
75	18898	7756
100	18861	7761
250	20016	7851
500	20027	7913
1000	20216	7912



Graf 6.5 Závislost kvality řešení na omezení dohlednosti

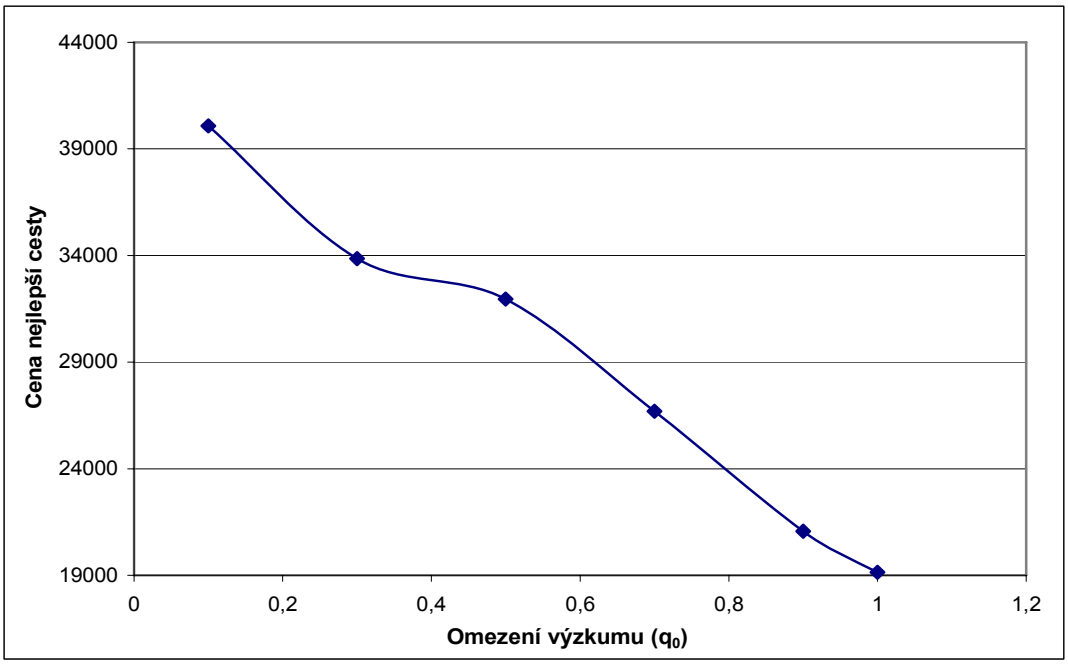
Z grafu je dobře patrné, že pro dané testovací soubory poskytoval algoritmus nejlepší výsledky při hodnotách okolo $\beta = 100$. Dalším zjištěným poznatkem je, že nastavování parametru není závislé na počtu prohledávaných měst.

6.6 OMEZENÍ VÝZKUMU

Pomocí parametru omezení výzkumu značeného jako q_0 lze nastavit, jak moc bude algoritmus upřednostňovat využívání znalostí o systému (ve formě ceny jednotlivých hran a hodnot feromonů na nich) na úkor výzkumu. Pro q_0 blízké jedné je vybráno jen lokálně optimální řešení. Naopak pro q_0 blízké nule jsou zkoumána všechna lokální řešení (ale lokálně optimálnímu řešení je stále přikládána větší váha).

Tabulka 6.7 Závislost kvality řešení na omezení výzkumu

Omezení výzkumu (q_0)	Cena nejlepší cesty
0,1	40080
0,3	33856
0,5	31959
0,7	26699
0,9	21065
1	19145



Graf 6.6 Závislost kvality řešení na omezení výzkumu

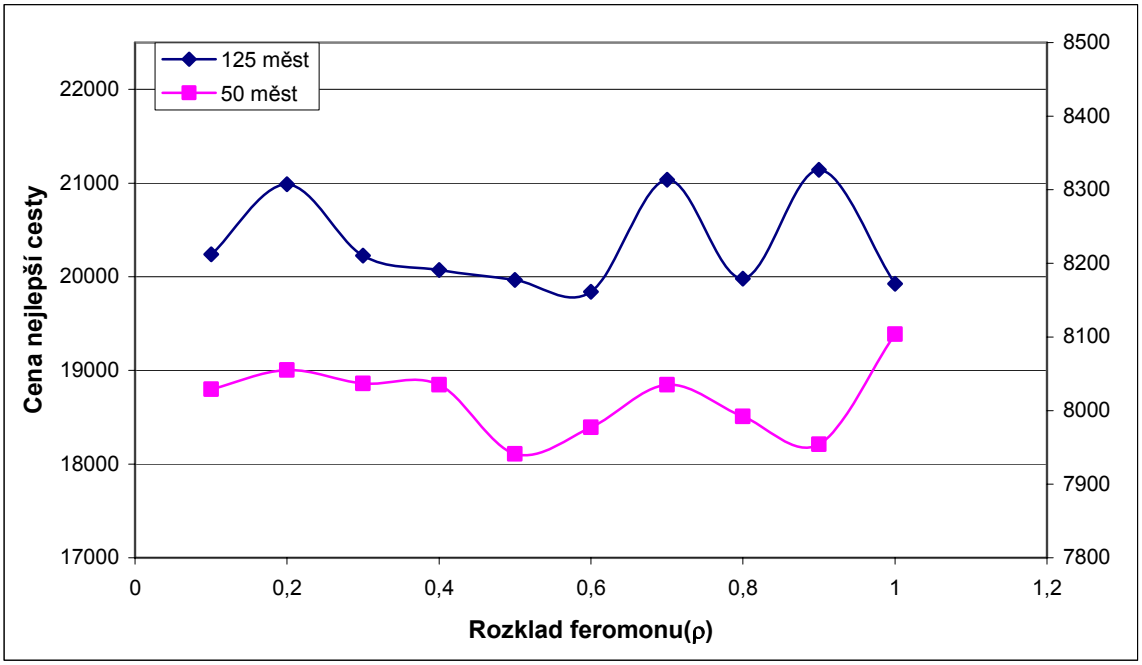
Při testování bylo zjištěno, že pro daný řešený problém je optimální nastavení parametru omezení dohlednosti blízké jedné.

6.7 ROZKLAD FEROMONU

Rozklad feromonu (ρ) je parametrem udávajícím míru samovolného odpaření (čili odstranění) feromonu z každé hrany v každé iteraci. Testování bylo prováděno na 125 i 50 městech.

Tabulka 6.8 Závislost kvality řešení na rozkladu feromonu

	Cena nejlepší cesty	
Rozklad feromonu(ρ)	125 měst	50 měst
0,1	20239	8029
0,2	20987	8055
0,3	20226	8037
0,4	20073	8035
0,5	19966	7941
0,6	19838	7977
0,7	21037	8035
0,8	19980	7992
0,9	21143	7954
1	19926	8104



Graf 6.7 Závislost kvality řešení na rozkladu feromonu

Ačkoli výsledek testování není zcela průkazný, lze tvrdit, že pro daný problém bylo nejlepším nastavením rozkladu feromonu hodnoty od 0,3 do 0,6. Závislost mezi nastavováním parametru a počtem prohledávaných měst nebyla nalezena.

7. ZÁVĚR

V první části práce byly shrnuty a popsány nejrozšířenější optimalizační algoritmy. Dále byl popsán vznik, metaheuristika a aplikace optimalizace pomocí mravenčí kolonie.

Byly naprogramovány tři optimalizační algoritmy k řešení problému obchodního cestujícího, a to Tabu Search, Simulated Annealing a ACO. Všechny tři algoritmy byly porovnány co do časové náročnosti zpracování a kvality produkovaného řešení.

Co do časové náročnosti výpočtu se nejlepším ukázal algoritmus zakázaného prohledávání (Tabu Search), který v tomto ohledu spolehlivě předčil oba zbývající algoritmy. Druhým nejlepším se ukázal algoritmus mravenčí kolonie (ACO) a nejhorším pak simulované žíhání (Simulated Annealing). Nebyl mezi nimi však tak propastný rozdíl jako mezi ACO a Tabu Search. Je nutno též poznamenat, že časová náročnost každého z algoritmů závisí na jeho řídicích parametrech (závislosti časové náročnosti na parametrech jsou dobře vidět z grafů v dané kapitole), a tudíž nelze tvrdit, že ACO je vždy rychlejší než Simulated Annealing.

Dále byla zkoumána kvalita řešení poskytovaného algoritmy. Nejprve byl proveden test, zda dokáží algoritmy najít nejlepší řešení jednoduchého problému. Tímto problémem bylo nalezení nejkratší cesty městy (body) rozmístěnými po obvodu kružnice. Tímto testem prošly úspěšně všechny algoritmy. Dále byly algoritmy testovány na reálných datech. Zde nejlepších výsledků dosáhl algoritmus ACO.

Poslední část práce byla věnována testování algoritmu ACO. Byl ověřován vliv jednotlivých řídicích parametrů na kvalitu řešení nalezeného algoritmem.

Pokud shrneme výsledky všech testů, pak nejlépe dopadl algoritmus ACO, protože dokázal poskytnout nejlepší řešení při relativně nevysoké výpočetní náročnosti, která sice byla podstatně větší než u Tabu Search, ale ten nedokázal poskytovat dostatečně dobrá řešení problému.

8. SEZNAM POUŽITÉ LITERATURY

[1] PaedDr. Mgr. Hashim Habiballa, PhD. : *Umělá Inteligence*. Dostupné na internetu: <www.volny.cz/habiballa/publ/umint.pdf>

[2] Prof. Ing. Václav Matoušek, CSc.: *Evoluční algoritmy a neuronové sítě.* Dostupné na internetu: <www.kiv.zcu.cz/studies/predmety/uir/texty/Chapter_09.pdf>

[3] Oleg Kovařík: *Ant Colony Optimization (ACO).* Dostupné na internetu: <http://ant-colony-optimization.no-ip.org/>

[4] Miloš Němec: *Optimalizace pomocí mravenčích kolonií.* Dostupné na internetu: <http://www.milosnemec.cz/clanek.php?78>

[5] Kolektiv autorů/Wikipedia: *Optimalization Algorithms.* Dostupné na internetu: <http://en.wikipedia.org/wiki/Category:Optimization_algorithms>

[6] Kolektiv autorů/CsWikipedia: *Genetický algoritmus.* Dostupné na internetu: <http://cs.wikipedia.org/wiki/Genetick%C3%BD_algoritmus>

[7] Ing. Ondřej Hodousek: *Algoritmy pro optimalizaci nehierarchické telekomunikační sítě.* Dostupné na internetu: <http://access.feld.cvut.cz/rservice.php?akce=tisk&cisloclanku=2005010702>

[8] Manuel Laguna: *Scatter search.* Dostupné na internetu: <http://leeds-faculty.colorado.edu/laguna/scattersearch.htm>

[9] Allan Rae: *Application-Specific Heterogenous Multiprocessor Synthesis Using Differential Evolution.* . Dostupné na internetu: <www.cs.nthu.edu.tw/~isss98/SLIDEs/3_1-Allen_Rae-new.ppt>

[10] Kolektiv autorů/CsWikipedia: *Genetický algoritmus.* Dostupné na internetu: <http://cs.wikipedia.org/wiki/Genetick%C3%BD_algoritmus>

[11] Pavel Pleva: *Problém obchodního cestujícího.* Dostupné na internetu: <http://nightmare.sh.cvut.cz/~pp/paa/tsp.html>

[12] Kolektiv autorů: *BOINC TSP.* Dostupné na internetu: <http://bob.myisland.as/tsp> a <https://svn.origo.ethz.ch/tsp/tsp/branches/mweltin/Instances/ATT/>

[13] Tomáš Kubeš: *Řešení problému obchodního cestujícího.* Dostupné na internetu: <http://www.tomaskubes.net/CVUT/cestujici.htm>

SEZNAM PŘÍLOH

Příloha A – CD-ROM

Přílohou diplomové práce je CD-ROM s následující adresářovou strukturou:

- Diplomová práce/: Elektronická verze diplomové práce a soubor metadat
- Zdrojové kódy/ACO/: Zdrojové kódy algoritmu Mravenčí kolonie
- Zdrojové kódy/SimulatedAnnealing/: Zdrojové kódy algoritmu Simulované žíhání
- Zdrojové kódy/TabuSearch /: Zdrojové kódy algoritmu Zakázané prohledávání
- Testovací data/: Datové soubory použité k testování vlastností algoritmů